Bachelorarbeit

Sascha Bartels

Automatisierung von Black Box Softwaretests
mit risikobasierter Priorisierung bei Eurogate
IT Services GmbH

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sascha Bartels

Automatisierung von Black Box Softwaretests mit risikobasierter Priorisierung bei Eurogate IT Services GmbH

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Bettina Buth
Zweitgutachter : Prof. Dr. Franz Korf
Abgegeben am 26.3.2009

Sascha Bartels

Thema der Bachelorarbeit

Automatisierung von Black Box Softwaretests mit risikobasierter Priorisierung bei Eurogate IT Services GmbH

Stichworte

Softwaretest, Black Box Test, Automatisierung, Regressionstest, Qualitätssicherung, HP Quality Center, QuickTest Professional, Squish

Kurzzusammenfassung

Thema dieser Arbeit ist die Vorgehensweise zur Einführung von automatisierten Black Box Softwaretests in einem betrieblichen Umfeld. Anhand von zwei unterschiedlichen Projekten werden Voraussetzungen und wichtige Schritte zur Implementierung beschrieben. Dies umfasst die Analyse der zu testenden Programme und Testprozesse, die Auswahl von Automatisierungstools sowie die konkrete Arbeit mit diesen Tools. Die Arbeit mit den Automatisierungstool HP QuickTest Professional wird mit Praxisbeispielen verdeutlicht.

Sascha Bartels

Title of the paper

Automation of Black Box Softwaretests with Risc-Based Priorization at Eurogate IT Services GmbH

Keywords

Software test, Black-Box Test, Test Automation, Regression Test, Quality Assurance, HP Quality Center, QuickTest Professional, Squish

Abstract

This thesis specifies an approach to the implementation of automatic black-box software tests in a business environment. Requirements and essential steps of an implementation are described on the basis of two different projects. This concept includes the analysis of the applications under test and the test process, the selection of automatic test tools as well as the practical work with these tools. The work with the automation tool HP QuickTest Professional is exemplified by practical examples.

Inhaltsverzeichnis

Abbildungsverzeichnis 6
1. Einleitung 7
1.1 Umfeld 7
1.2 Motivation 9
1.3 Aufbau 10
1.4 Abgrenzung 11
1.5 Einleitende Projektbeschreibungen 11
1.5.1 COIN 11
1.5.2 TOPX 12
1.6 Begriffe 12
2 Testgrundlagen 15
2.1 Teststufen 15
2.1.1 Komponententest 16
2.1.2 Integrationstest 17
2.1.3 Systemtest 18
2.1.4 Abnahmetest 19
2.2 Klassifizierung von Prüftechniken 20
2.2.1 Black Box und White Box Prüftechniken 20
2.2.2 Statische und dynamische Prüftechniken 21
2.3 Regressionstests 22
2.4 Anforderungen 23
2.5 Testfallerstellung 24
2.6 Risikoanalyse und Priorisierung von Testfällen 25
3 Testautomatisierung 27
3.1 Automatisierter Vergleich 27
3.2 Automatisierte Vor- und Nachbereitung 28
3.3 Erstellung von wartbaren Tests 28
3.4 Metriken 29
3.5 Was kann und sollte automatisiert werden? 32
4 Projektbeschreibungen 34
4.1 COIN 34
4.1.1 Technische Beschreibung 34
4.1.2 Funktionalität von COIN 34
4.1.3 Projektbeschreibung 35
4.1.4 Testanforderung 36
4.2 TOPX 37
4.2.1 Technische Beschreibung 37
4.2.2 Aufgaben von TOPX 38
4.2.3 Projektbeschreibung 40
4.2.4 Testanforderung 41
5 Toolauswahl und Bewertung 42
5.1 Hinweise zur Toolauswahl 42
5.1.1 Anforderungen an eine Einführung von automatisierten Tests 42
5.1.2 Auswahlkriterien 43
5.2 HP Quality Center 43
5.2.1 Releases 44
5.2.2 Requirements 45
5.2.3 Business Components 47

5.2.4 Test Plan ... 48
5.2.5 Test Lab ... 48
5.2.6 Defects ... 50
5.2.7 Zusammenfassung ... 51
5.3 Auswahl der Automatisierungstools ... 52
5.4 HP Quicktest Professional ... 54
5.4.1 Skripterstellung ... 54
5.4.2 Verifikation der Ergebnisse ... 55
5.4.3 Datengetriebener Test ... 57
5.4.4 Zusammenfassung ... 57
5.5 Redstone Software Eggplant ... 58
5.5.1 Skripterstellung ... 58
5.5.2 Skriptsprache ... 58
5.5.3 Bilderkennung ... 59
5.5.4 Texterkennung ... 59
5.5.5 Zusammenfassung ... 60
5.6 Froglogic Squish ... 61
5.6.1 Skripterstellung ... 61
5.6.2 Verifikation der Ergebnisse ... 61
5.6.3 Zusammenfassung ... 63
5.7 Ergebnis der Toolauswahl ... 64
6 Automatisierung der Tests für die COIN-Migration ... 66
6.1 Testkonzept ... 66
6.2 Automatisiertes Aufrufen der Funktionen ... 66
6.3 Automatisiertes Überprüfen der Maskeninhalte ... 70
6.3.1 Erstellung von Referenzdaten ... 71
6.3.2 Vergleich mit Referenzdaten ... 72
6.3.3 Analyse der Testergebnisse ... 74
6.4 Wirtschaftlichkeit der Automatisierung ... 75
6.5 Synthetische Daten ... 76
6.5.1 Verwendung von synthetischen Daten ... 77
6.5.2 Erzeugung von synthetischen Daten in COIN ... 77
6.6 Testen von Geschäftsprozessen ... 79
6.7 Aktueller Stand und weiteres Vorgehen ... 80
7 Automatisierung der TOPX Tests ... 81
7.1 Ist-Zustand ... 81
7.1.1 Testfallermittlung ... 81
7.1.2 Testdokumentation ... 82
7.1.3 Testpriorisierung ... 82
7.1.4 Testausführung ... 82
7.2 Zielzustand ... 82
7.3 Skizzierung einer möglichen Umsetzung ... 83
8 Zusammenfassung und Ausblick ... 87
Quellenverzeichnis ... 90
Glossar ... 92

Abbildungsverzeichnis

Abbildung 1-1: Eurogate Container Terminal Hamburg 8
Abbildung 2-1: allgemeines V-Modell 15
Abbildung 4-1: COIN-Hauptmenü 35
Abbildung 4-2: Aufgereihte Container mit Van Carriern 38
Abbildung 4-3: Lade- und Löschvorgang am CT Bremerhaven 39
Abbildung 4-4: Verladung von Stückgut 40
Abbildung 5-1: HP Quality Center Graphen Testfallüberdeckung 45
Abbildung 5-2: HP Quality Center Requirements 46
Abbildung 5-3: HP Quality Center Risikoanalyse 47
Abbildung 5-4: HP Quality Center Test Plan 48
Abbildung 5-5: HP Quality Center Test Lab Execution Flow 49
Abbildung 5-6: HP Quality Center Test Lab manuelle Testausführung 50
Abbildung 5-7: HP Quality Center Defects 51
Abbildung 5-8: HP QuickTest Professional Action im Keyword View 54
Abbildung 5-9: HP QuickTest Professional Checkpoint 56
Abbildung 5-10: Eggplant Hotspot setzen 59
Abbildung 5-11: Squish Adressbuch Beispiel 62
Abbildung 6-1: COIN-Fenster mit Funktionsaufruf 67
Abbildung 6-2: COIN-Maske 362 – Containeranzeige 68
Abbildung 6-3: Data Table in QuickTest Professional 68
Abbildung 6-4: QuickTest Professional Data Table "Global" 69
Abbildung 6-5: HP QC Reporter 73
Abbildung 6-6: Geschäftsprozess Exportcontainer erfassen 78
Abbildung 6-7: HP QTP Data Table Container anlegen 78
Abbildung 6-8: HP QTP Data Table globales Datenblatt 79
Abbildung 7-1: TOPX mit geöffneter Schiffsansicht 84

1. Einleitung

Diese Arbeit entstand im Rahmen der Gründung des Bereichs „Testmanagement“ in der Eurogate IT Services GmbH, einem Tochterunternehmen der Eurogate GmbH & Co KGaA. Im Kapitel 1.1 Umfeld werden diese Unternehmen kurz vorgestellt.

Welche Ziele diese Arbeit hat und welche nicht, wird in Kapitel 1.2 Motivation und Kapitel 1.4 Abgrenzung dargestellt. Anschließend folgt eine Begriffsklärung zu den wichtigsten Ausdrücken, die während der Arbeit mit Softwaretests verwendet werden.

Jede Branche hat ihre eigene Fachsprache. Gerade die Begriffe in einem Container Terminal sind nicht immer leicht verständlich. Aus diesem Grund befindet sich im Anhang ein Glossar mit den wichtigsten Begriffen, die zum Verständnis der Arbeitsabläufe nötig sind. Um den Lesefluss nicht zu stören, werden diese Begriffe beim ersten Vorkommen im Text kurz erläutert.

1.1 Umfeld

Die Eurogate GmbH & Co. KGaA ist mit einem Umschlag von 13,9 Mio. TEU (20 Fuß ISO-Container; „Twenty feet Equivalent Unit“) im Jahre 2007 und neun Terminal-Standorten Europas größter Container-Terminal-Betreiber. Zusammen mit der Contship Italia S.p.A. werden Seeterminals an der Nordsee, im Mittelmeerraum und am Atlantik mit Verbindungen ins europäische Hinterland betrieben. Neben dem Containerumschlag werden diverse Dienstleistungen angeboten, von cargomodalen Services (Transportdienstleistungen) über Container-Depot bis Container-Wartung und -Reparatur, sowie intermodaler Transport (Transportkette mit verschiedenen Verkehrsträgern) und Logistik-Management, IT-Logistik-Lösungen und spezialisierten Ingenieurleistungen. In Deutschland werden ca. 4.500 Mitarbeiter beschäftigt.
[vgl. Eurogate2008-1]

Der Container Terminal Hamburg (CTH) liegt in Hamburg-Waltershof. Dort befinden sich zurzeit sechs Groß-Schiffsliegeplätze mit 21 Containerbrücken. 2007 wurden hier 2,9 Millionen TEU umgeschlagen.
[vgl. Eurogate2008-2]

Abbildung 1-1: Eurogate Container Terminal Hamburg
[Eurogate2007, S. 1]

Die Eurogate IT Services GmbH (EG ITS) bietet IT-Dienstleistungen für den Containerumschlag sowie für cargomodale und intermodale Logistik an. Das Unternehmen ist für die gesamte IT von Eurogate verantwortlich. Dieses Aufgabenfeld umfasst die Bereitstellung der Infrastruktur mit ca. 200 Servern, auf denen verschiedene Betriebssysteme zum Einsatz kommen, das Emailsystem, die Desktoprechner und Drucker sowie die Wartung und den Support für die Systeme. Es werden viele verschiedene Softwaresysteme verwaltet, weiterentwickelt und in Projektarbeiten ergänzt. Hierbei handelt es sich um Programme zur Containerverwaltung, Koordination der operativen Abläufe auf dem Yard, ERP-Systeme (SAP), Port Security, E-Business und einige andere Systeme. Damit diese Systeme mit allen wichtigen Daten versorgt werden, bietet EG ITS EDI-Schnittstellen (Electronic Data Interchange) für Kunden, Reeder und Behörden an. Außerdem werden die Internet- und Intranetauftritte des gesamten Konzerns mittels eines CMS (Content Management Systems) betreut. Zu diesen Online-Anwendungen kann man auch das „Infogate“ zählen. Es ist ein Internet-Portal, in dem Daten für Kunden bereitgestellt werden. Der Kunde kann darüber Informationen über den Schiffsverkehr am Terminal erhalten sowie Lösch- und Ladelisten und wichtige Informationen über seine Container.
[vgl. Eurogate2008-3]

1.2 Motivation

Software muss getestet werden, das ist keine neue Erkenntnis. Bekannt sind vielleicht solche verheerende Softwarefehler wie jener, der die Venussonde Mariner 1 im Jahre 1962 zur kontrollierten Selbstzerstörung veranlasste. Der Schaden belief sich auf ca. 18,5 Mio. $.Ein ähnlicher Vorfall zerstörte 1996 die Ariane 5, deren Entwicklungskosten innerhalb von 10 Jahren 5,5 Mrd. $ betrugen.
[vgl. Vigenschow2005 S. 9-12]

Mit Sicherheit hat sich schon jeder über Fehler in kommerziellen und nicht-kommerziellen Softwareprodukten geärgert. Im betrieblichen Umfeld können solche Fehler immense finanzielle Verluste bedeuten. Das Testen von Individualsoftware ist nicht nur dem Hersteller dieser vorbehalten, auch der Kunde, der eine solche Software in Auftrag gegeben hat, ist spätestens beim Abnahmetest dazu angehalten, das Produkt ausreichend zu überprüfen. Wird das Softwareprodukt fälschlicherweise für einsatzfähig erklärt, können - je nach Vertrag - weitere Kosten auf den Kunden zu kommen, um diese Fehler zu beheben.

Softwarefehler verursachen große Kosten und meistens sind diese Kosten umso größer, je später sie entdeckt werden. Am fatalsten sind also die Fehler, die erst bei der Nutzung des Softwareprodukts auffallen.
[vgl. Liggesmeyer2002, S. 29]

Auch bei Eurogate kam es zu einer solchen Situation. Im Juni 2008 wurde am Containerterminal Hamburg der Firma Eurogate ein Teil der Terminal Operation Software (TOS) ersetzt. Das neue Modul TOPX (Terminal Operation Package XWindow) ist für die Schiff-, Platz- und Geräteplanung zuständig. Doch bereits am ersten Tag traten verschiedene Probleme mit TOPX auf. Diese Probleme führten dazu, dass die LKW-Abfertigung nur äußerst schleppend durchgeführt werden konnte. Da bei Eurogate täglich bis zu 3500 LKW abgefertigt werden, entstand ein Verkehrsstau von 25 km Länge.
[vgl. Gaßdorf2008]

Auch das Testen verursacht Kosten. Gründliche Tests von komplexen Softwaresystemen sind zeitintensiv und personalaufwendig. Also liegt es nahe, dass Softwaretests optimiert werden sollen. Ein Mittel zur Optimierung ist die Einführung von automatisierten Softwaretests. Meistens werden sehr hohe Erwartungen an den durch Automatisierung erreichten Vorteil gestellt. Und sehr häufig ist es nicht möglich, diese Erwartungen zu erfüllen. Ein schlecht organisierter Testprozess wird auch durch Automatisierung nicht zu einem besseren Prozess. „Automating chaos just gives faster chaos.“ [Fewster1999, S.11]

Tests zu automatisieren ist eine Idee, die es schon länger gibt. Entsprechend gibt es umfangreiche Literatur zu diesem Themenkomplex. Hierbei handelt es sich meistens um generelle Vorgehensweisen und Hinweise darauf, wo Probleme entstehen können. Wie die tatsächliche Umsetzung einer solchen Automatisierung erfolgen kann, ist oftmals nicht beschrieben. Dies mag daran liegen, dass es offensichtlich keine feste Vorgehensweise gibt. Jedes Softwareprojekt ist anders und wird in einer anderen Umgebung eingesetzt. Unterschiedliche Programmiersprachen, verschiedene Plattformen, andere Hardwarearchitekturen, all diese Faktoren haben Einfluss darauf, wie eine Automatisierung realisiert werden kann. Auch die Automatisierung kann verschiedene Zielsetzungen haben. So

kann entweder die Menge an Tests gleich bleiben, aber in kürzerer Zeit ausgeführt werden, oder man möchte in der gleichen Zeit, die für manuelle Tests benötigt wurde, wesentlich mehr und somit gründlicher testen. Daraus würden sich selbst für identische Projekte schon zwei unterschiedliche Ansätze ergeben. Um diese Brücke zwischen Theorie und Praxis zu bauen, werden in einigen Fachbüchern Erfahrungsberichte über unterschiedliche betriebliche Projekte abgedruckt. Wie zum Beispiel in [Fewster1999]. Auch in dieser Arbeit werden zwei recht unterschiedliche Projekte aus der Praxis bearbeitet.

Die grundsätzliche Aufgabenstellung dieser Arbeit bei Eurogate ITS ist die Analyse der aktuellen Projekte in Hinblick auf konkrete Möglichkeiten der Unterstützung von Softwaretests mit automatisierten Tests oder sogar eine vollständige Automatisierung. Diese Arbeit entstand während der im Juli 2008 neu eingerichtete Bereich „Testmanagement“ seine Arbeit aufnahm. Ziel dieses Bereiches ist es, die bisher extern durchgeführten Softwaretests im Hause selbst zu organisieren. Zwei Projekte, die auch in dieser Arbeit vorgestellt und behandelt werden, haben hier die höchste Priorität.

Bei dem einen Projekt handelt es sich um eine 1:1 Migration eines COBOL Programms, das zum ersten Mal 1983 in Betrieb genommen wurde. Dieses Programm soll durch eine Codekonvertierung auf ein moderneres COBOL gebracht werden und auf einem Oracle-Datenbanksystem aufsetzen. Die Zielsetzung des Tests ist zu zeigen, ob das System nach der Konvertierung immer noch die gleichen Anforderungen erfüllt.

Das zweite Projekt hat das weiter oben angesprochene TOPX zum Thema. Für TOPX werden weiterhin Patches und Releases entwickelt, die getestet werden müssen. Diese Tests werden von dem Bereich „Testmanagement“ geleitet und weiterhin von dem externen Dienstleister unterstützt, der aber auf lange Sicht abgelöst werden soll. Man möchte erreichen, dass die Tests komplett von Eurogate ITS übernommen werden. Aktuell werden nur die Änderungen, die die einzelnen Patches beinhalten, überprüft. Es werden keine speziellen Tests durchgeführt, die gezielt Fehler aufdecken sollen, die in diesen neuen Versionen hinzu gekommen sind. Das „Traumziel“ wäre die Möglichkeit, einen kompletten Systemtest automatisch laufen lassen zu können. Man müsste dann nur noch ein neues Release einspielen und anschließend den automatischen Test überprüfen lassen, ob irgendwo ein Fehler neu aufgetreten ist. Dieses Ziel ist nicht ohne größeren Aufwand zu erreichen, sofern es überhaupt möglich ist. Trotzdem sollen die Testabläufe optimiert werden, eben auch durch die Einführung automatischer Tests.

1.3 Aufbau

Diese Arbeit beginnt mit einem einführenden Teil über den Themenkomplex „Softwaretests“ mit einem Schwerpunkt in der Testautomatisierung. Anschließend wird die Auswahl von Tools zur Unterstützung von automatisierten Tests diskutiert. Einer Beschreibung der allgemeinen Vorgehensweise folgt die Darstellung der konkreten Umsetzung bei Eurogate mit Bezug auf die beiden hier herausgearbeiteten Projekte.

Im Praxisteil dieser Arbeit wird gezeigt, wie mit den gewählten Hilfsmitteln eine Automatisierung in den hier vorgestellten Projekten implementiert werden kann. Im Abschluss werden die hierbei gewonnenen Erkenntnisse zusammen getragen und um einen Ausblick auf eine weitere Vorgehensweise ergänzt. Der Schwerpunkt liegt hier zum einen auf

den Problemen, die die Umsetzung der Automatisierung erschweren, und zum anderen auf den Vorschlägen, wie sie umgangen oder behoben werden können.

1.4 Abgrenzung

Sowohl die Neuentwicklung von COIN, als auch die Entwicklung von TOPX wird außer Haus von externen Dienstleistern durchgeführt. Diese Firmen sind somit auch für die Durchführung von Softwaretests auf unterschiedlichen Entwicklungsstufen verantwortlich. Der Quellcode dieser Produkte liegt nicht vor. Somit können von ITS nur Black Box Tests, also Tests ohne Kenntnis des Quellcodes, ausgeführt werden.

Weitere Testmethoden und Teststufen werden zwar im theoretischen Teil erläutert, aber im praktischen Teil nicht weiter vertieft.

Wenn im Rahmen dieser Arbeit von einer Qualitätsverbesserung die Rede ist, dann bezieht sich dies lediglich auf den Testprozess, nicht aber auf die Software, die getestet wird. Eine Verbesserung wäre ein Zeitgewinn bei der Durchführung der Tests oder eine bessere Fehlererkennung, zum Beispiel durch eine höhere Testabdeckung der Anforderungen. Die erkannten Fehler sollen dann an die Entwickler weitergegeben werden, damit diese sie dann (hoffentlich) beheben. Dies führt natürlich zu einer Verbesserung der Qualität der Software, wird aber in dieser Arbeit nicht behandelt.

Einer vollständigen Automatisierung dieser Projekte stehen verschiedene Faktoren entgegen. Die Komplexität dieser Aufgabenstellen erlaubt nur die Darstellung eines exemplarischen Vorgehens unter Herausarbeitung ausgesuchter Testfälle. Es wird kein vollständiger Test dokumentiert, weder ein manueller noch ein automatisierter.

Das gesamte Softwaresystem bei Eurogate besteht aus diversen Programmen mit einer Vielzahl an Schnittstellen. Aufgrund dieser Komplexität werden nur die beiden großen Programmteile COIN und TOPX näher beschrieben. Auf eine detaillierte Aufstellung aller Umsysteme und Schnittstellen wird verzichtet.

1.5 Einleitende Projektbeschreibungen

Diese Beschreibungen der beiden Softwareprojekte, die in dieser Arbeit aus dem Blickwinkel des automatisierten Testens betrachtet werden, dienen zu einer ersten Orientierung und um einige Fakten herauszustellen. Eine genauere Beschreibung erfolgt in den Kapiteln Automatisierung der Tests für die COIN-Migration und Automatisierung von TOPX, die sich vor allem auf die Umsetzung der eher theoretischen Teile dieser Arbeit auf die Praxis beziehen.

1.5.1 COIN

COIN ist die Abkürzung für Container Information. In diesem System werden alle Daten über Container erfasst, die per Schiff, LKW oder Bahn angeliefert werden. Diese Daten enthalten unter anderem sämtliche Details über die Container wie Größe, Gewicht, Zielort, Ankunftszeiten etc. COIN wurde 1983 in Betrieb genommen und ist in COBOL

programmiert. Seit der Inbetriebnahme wurde es regelmäßig weiterentwickelt. COIN wird mittels der Terminalemulation Exceed (www.hummingbird.com) auf Windows-Rechnern ausgeführt. Die Oberfläche ist rein textbasiert und wird durch die Eingabe von Funktionen, Schlüsselwerten und bestimmten Aktionen bedient.

Die Datenhaltung erfolgt auf einem hierarchischen Datenbanksystem auf einem Unisys Host. Für diesen Host läuft der technische Support in naher Zukunft aus. Aus diesem Grunde wurde eine Modernisierung von COIN beschlossen. Ein externer Dienstleister soll die Codes auf ein moderneres Micro Focus COBOL konvertieren und die Datenhaltung auf eine Oracle-Datenbank portieren. Für dieses Migrationsprojekt soll ein Testkonzept aufgestellt werden, das an sinnvollen Punkten durch automatisierte Prozesse optimiert wird.

Die Tatsache, dass ein Neu- und ein Altsystem mit den gleichen Anforderungen parallel existieren sollen, ist nicht alltäglich, bietet aber für den Test Möglichkeiten, die bei einer einfachen Entwicklung nicht gegeben sind. Dieser Test ist trotzdem kein Einzelfall, da solche Migrationsprojekte in der Praxis häufiger vorkommen.

1.5.2 TOPX

Bei TOPX handelt es sich um eine operative Planungssoftware für die Abläufe an Container Terminals. Es ist eine Standardsoftware, die an die spezifischen Gegebenheiten der unterschiedlichen Terminals und Betreiber angepasst wird. TOPX wurde 2008 bei Eurogate eingeführt und wird durch regelmäßige Patches und Releases weiter entwickelt. Es basiert auf dem QT-Framework von Trolltech (www.trolltech.org) und wird auf einem SUN Solaris System ausgeführt. Der Zugriff von den Windows-Rechnern bei Eurogate erfolgt ebenfalls über Exceed. Mit der Gründung des Fachbereichs Testmangement sollen die Testverantwortung stückweise von einem externen Dienstleister übernommen werden. Im Rahmen dieser Arbeit soll erarbeitet werden, wie die benötigten Tests mit einer Automatisierung unterstützt werden können. Das optimale Wunschziel wäre ein komplett automatisierter Regressionstest. Welche Voraussetzungen dabei erfüllt werden müssen und ob dies in der Praxis durchführbar ist, wird in dieser Arbeit ermittelt.

1.6 Begriffe

Folgende Begriffe treten im Zusammenhang mit Softwaretests immer wieder auf und werden deswegen hier einmal erläutert: (Alle Begriffe und Erklärungen sind [Linz2005] entnommen.)

Fehler: 1. Oberbegriff für Fehlerhandlung, Fehlerzustand, Fehlerwirkung.
2. Nichterfüllung einer festgelegten Anforderung.

Fehlermaskierung: Ein vorhandener Fehlerzustand wird durch einen oder mehrere andere Fehlerzustände kompensiert, so dass dieser Fehlerzustand keine Fehlerwirkung hervorruft.

Fehlerwirkung: 1. Wirkung eines Fehlerzustands, die bei der Ausführung des Testobjekts nach außen in Erscheinung tritt.
2. Abweichung zwischen Sollwert und Istwert.

Fehlerzustand: 1. Inkorrektes Teilprogramm, Anweisung oder Datendefinition, die Ursache für eine Fehlerwirkung ist.
2. Zustand eines Softwareprodukts oder einer seiner Komponenten, der unter spezifischen Bedingungen eine geforderte Funktion des Produkts beeinträchtigen kann bzw. zu einer Fehlerwirkung.

Fehlhandlung: 1. Die menschliche Handlung (des Entwicklers), die zu einem Fehlerzustand in der Software führt.
2. Eine menschliche Handlung (des Anwenders), die ein unerwünschtes Ergebnis (Fehlerwirkung) zur Folge hat (Fehlbedienung).
3. Unwissentlich, versehentlich oder absichtlich ausgeführte Handlung oder Unterlassung, die unter gegebenen Umständen dazu führt, dass eine geforderte Funktion eines Produkts beeinträchtigt wird.

Funktionaler Test: Dynamischer Test, bei dem die Testfälle unter Verwendung der funktionalen Spezifikation des Testobjekts hergeleitet werden und die Vollständigkeit der Prüfung (Überdeckungsgrad) anhand der funktionalen Spezifikation bewertet wird.

Mangel: Eine gestellte Anforderung oder berechtigte Erwartung wird nicht erfüllt.

Nicht funktionaler Test: Prüfung der nicht funktionalen Anforderungen. Nach ISO 9126: Zuverlässigkeit, Benutzbarkeit, Effizienz.

Testendekriterium: Kriterien, die vorab festgelegt werden und erfüllt sein müssen, um eine (Test-)Aktivität als abgeschlossen bewerten zu können.

Testaufwand: Bedarf an Ressourcen für den Testprozess.

Testauswertung: Anhand der Testprotokolle wird ermittelt, ob Fehlerwirkungen vorliegen; ggf. wird eine Einteilung in Fehlerklassen vorgenommen.

Testorakel: Informationsquelle zur Ermittlung der jeweiligen Sollergebnisse eines Testfalls (z.B. die Anforderungsspezifikation).

Teststufe: Eine Teststufe ist eine Gruppe von Testaktivitäten, die gemeinsam ausgeführt und verwaltet werden. Zuständigkeiten in einem Projekt können sich auf Teststufen beziehen. Beispiele sind: Komponententest, Integrationstest, Systemtest (nach allg. V-Modell).

Überdeckungsgrad: Kriterium zur Beendigung des Tests, unterschiedlich je nach Testmethode, meist durch Werkzeuge ermittelt.

Es gibt viele Bezeichnungen für unterschiedliche Arten von Softwaretests. Nachfolgend sind einige Kategorien dargestellt, nach denen Testarten benannt werden können. Nicht jeder Begriff beschreibt eine völlig andere Art von Test, ausschlaggebend ist der Blickwinkel, aus dem die Testarbeit betrachtet wird.

1. Testziel: Zweck des Tests, z.B. Lasttest

2. Testmethode: Methode, die zur Spezifikation oder Durchführung des Tests eingesetzt wird, z.B. geschäftsprozess-basierter Test

3. Testobjekt: Art des Testobjekts, z.B. GUI-Test

4. Teststufe: Die Stufe des verwendeten Vorgehensmodells, z.B. Systemtest

5. Testperson: Der Personenkreis, der den Test durchführt, z.B. Entwicklertest

6. Testumfang: z.B. partieller Regressionstest, Volltest

[vgl. Linz2005, S. 10]

2 Testgrundlagen

In diesem Kapitel werden einige Grundlagen des Softwaretestens näher erläutert. Es wird ein Überblick über die gesamte Thematik gegeben, damit im späteren Verlauf der Arbeit die Möglichkeit besteht, die Tests entsprechend einzuordnen.

In Kapitel 2.1 wird der Testverlauf begleitend zu einem Vorgehensmodell, hier dem V-Modell, vorgestellt. Kapitel 2.2 beschäftigt sich mit verschiedenen Prüfmethoden und deren Klassifizierung, während in Kapitel 2.3 speziell auf die Regressionstests eingegangen wird. Kapitel 2.4 geht dann näher auf die Testautomatisierung ein, die einen zentralen Punkt dieser Arbeit bildet.

2.1 Teststufen

Das allgemeine V-Modell verbindet die Entwicklungsarbeiten mit den dazugehörigen Testaktivitäten. Auf der linken Seite der Modelldarstellung finden sich die verschiedenen Entwicklungsstufen, auf der rechten Seite stehen die zugehörigen Testarbeiten.

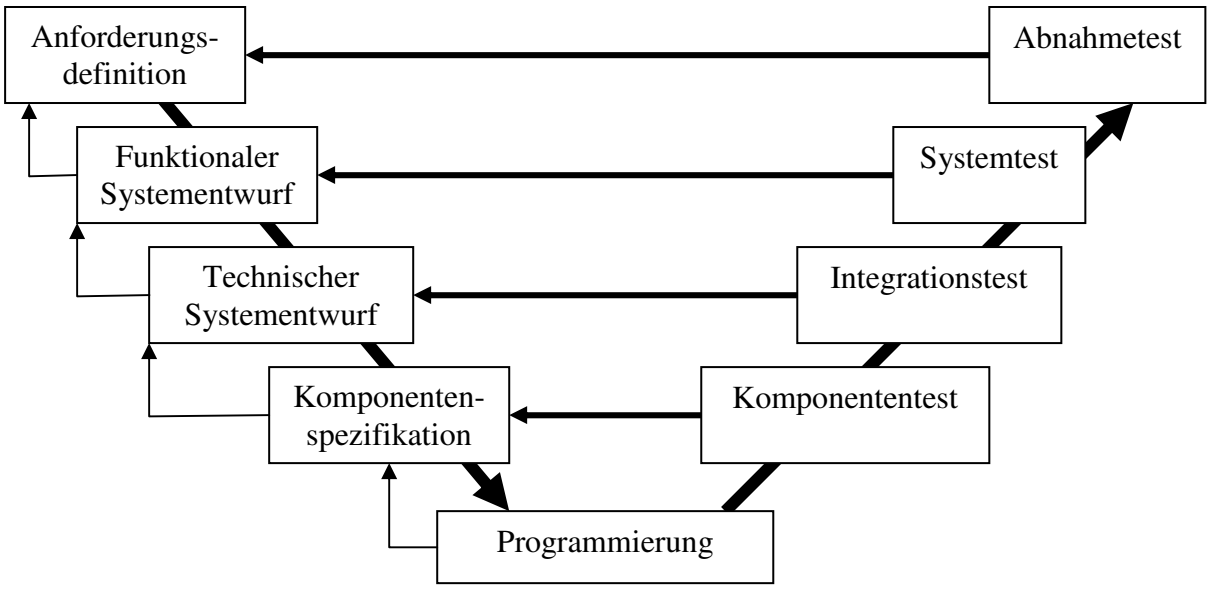


Abbildung 2-1: allgemeines V-Modell

Auf der linken Seite beginnen die konstruktiven Aktivitäten mit der Anforderungsdefinition. Hier werden die Anforderungen des Auftragsgebers spezifiziert. Im funktionalen Systementwurf werden die Anforderungen auf Funktionen des neuen Systems abgebildet. Die technische Realisierung, wie die Definition der Schnittstellen zur Systemumwelt und das Festlegen der Systemarchitektur, erfolgt in dem technischen Systementwurf. In der Komponentenspezifikation werden die Aufgaben, der innere Aufbau und die Schnittstellen

jedes Teilsystems festgehalten. Erst danach beginnt die eigentliche Programmierung, in der die spezifizierten Teile in einer Programmiersprache implementiert werden.
[vgl. Linz2005, S. 40]

Jeder dieser Entwicklungsstufen sind Teststufen auf der rechten Seite der Darstellung zugeordnet. Diese werden im Folgenden näher erläutert.

2.1.1 Komponententest

Der Komponententest ist der Test auf der untersten Stufe der Entwicklung. Er ist der erste systematische Test, dem die neu programmierten Module unterzogen werden. Man kann hier noch zwischen einem Modultest und einem Klassentest unterscheiden. Dies ist abhängig von der eingesetzten Programmiersprache. Bei prozeduralen Sprachen spricht man vom Modultest, da diese in Module aufgeteilt sind, während der Klassentest das Äquivalent zu den objektorientierten Sprachen bildet.

Beim Komponententest wird darauf geachtet, dass jede Komponente einzeln betrachtet wird. Es werden mögliche komponentenexterne Fehlerquellen ausgeschlossen, so dass eine gegebenenfalls auftretende Fehlerwirkung dieser einen getesteten Komponente zugeordnet werden kann. Ein Komponententest benötigt für gewöhnlich einen Testtreiber. Ein Testtreiber ist ein Programm, das die zu testenden Schnittstellen der Komponente aufruft und den zurückerhaltenen Wert auf Richtigkeit überprüft. Dieser Testtreiber kann natürlich weitere sinnvolle Funktionalitäten enthalten wie zum Beispiel eine Protokollierung der Ergebnisse.

Da der Komponententest entwicklungsnah ist und für die Erstellung eines Testtreibers Entwicklungs-Know-how benötigt wird, werden die Komponententests meistens von den Entwicklern selbst durchgeführt. Deswegen wird auch vom Entwicklertest gesprochen.

Natürlich ist es nicht unbedingt vorteilhaft, wenn die Entwickler ihren Code selber testen, meistens ist es eher hinderlich. Viele Entwickler konzentrieren sich lieber auf das Programmieren und weniger auf das Testen, was zu oberflächlichem Testen führen kann. Verstärkt wird dieser Effekt dadurch, dass den eigenen Programmierarbeiten für gewöhnlich mit einem großen Optimismus entgegengetreten wird.
[vgl. Linz2005]

Zwischen Modultests und Klassentests gibt es große Unterschiede. Bei prozeduralen Modulen wird Wert darauf gelegt, die Schnittstellen zu testen und möglichst viele logische Verzweigungen abzudecken. Mit der Größe der Module steigt die Komplexität der Ablauflogik, und somit wächst auch die Anzahl der möglichen Pfade vom Eingang des Moduls bis zum Ausgang. Oft werden diese Module so groß und die Schnittstellen so komplex, dass ein angemessener Modultest gar nicht durchführbar ist.
[vgl. Sneed2002, S. 159-160]

Bei Klassentests sieht die Problematik anders aus. Hier findet man eher kleine Schnittstellen und kleinere Methoden mit einer weniger komplexen Ablauflogik. Allerdings finden viele Aufrufe fremder Methoden auch aus anderen Klassen statt. Es gibt Abhängigkeiten zwischen

den Klassen, die auch durch Vererbung entstehen. Somit wird es schwer, die Klassen isoliert zu testen, was aber eine Anforderung des Komponententests ist.
[vgl. Sneed2002, S. 160-162]

Probleme beim Testen von Klassen entstehen aus den eigentlichen Vorteilen der objektorientierten Programmierung: Vererbung, Polymorphie, Überladen von Parametern und Wiederverwendung fremder Klassen.
[vgl. Sneed2002, S. 164]

Zusammenfassend lässt sich sagen, dass die Hürden beim Modultest in der Komplexität der Module liegen, während beim Klassentest die externen Abhängigkeiten den Testvorgang erschweren.

2.1.2 Integrationstest

Der Integrationstest schließt an den Komponententest an und setzt voraus, dass die einzelnen Komponenten bereits ausreichend getestet und gegebenenfalls korrigiert wurden. Anschließend werden verschiedene Komponenten zu einem Teilsystem zusammengesetzt. Ziel ist es nun, das Zusammenspiel dieser Komponenten und deren Schnittstellen zu testen.
[vgl. Linz2005, S. 50]

Fehler in den Schnittstellen, also inkonsistente Parameterlisten oder Rückgabewerte, fallen sofort beim Zusammensetzen von Komponenten auf. Diese Fehler sind sehr leicht aufzudecken, schwieriger wird es bei Problemen, die sich erst im dynamischen Test, also beim Ausführen des Codes, zeigen. Mögliche Fehlerzustände, die auftreten können, sind folgende:

- Es werden keine oder syntaktisch falsche Daten zwischen den Komponenten ausgetauscht, so dass es zu Fehlern oder Abstürzen kommt.
- Übergebene Daten werden falsch interpretiert.
- Timing-Probleme oder Kapazitätsprobleme können auftreten. Daten kommen zu spät an, oder es werden zu viele Daten in zu kurzer Zeit gesendet bzw. empfangen.

Alle diese Fehler fallen in einem Komponententest nicht auf, da sie ausschließlich durch Wechselwirkungen mit mehreren Komponenten entstehen.
[vgl. Linz2005, S. 53-54]

Der Integrationstest von größeren objektorientierten Anwendungen kann im Allgemeinen in drei Stufen unterteilt werden. Klassenintegration, Komponentenintegration und Schichtenintegration. Bei der Klassenintegration werden einzelne Klassen zusammengebunden, die zu einer Komponente gehören, und getestet. Wird nur eine kleine Applikation betrachtet, so ist an dieser Stelle der Integrationstest bereits abgeschlossen. Größere Anwendungen können in einer mehrschichtigen Architektur aufgebaut sein. Hierbei kann es sich um eine Client- und eine Serverschicht handeln oder um Architekturen mit drei und mehr Schichten. Werden alle Komponenten, die zu einer Anwendungsschicht gehören, nach und nach zusammengesetzt und getestet, spricht man von einer Komponentenintegration. Hier werden die Beziehungen zwischen den Komponenten sowie sämtliche Effekte, die durch das Zusammensetzten auftreten, getestet. Die dritte Stufe ist die

Schichtenintegration. Es werden verschiedene Anwendungsschichten zusammengeführt, wie zum Beispiel Client- und Serverschicht.
[vgl. Sneed2002, S. 195-197]

2.1.3 Systemtest

Bei der dritten Teststufe wird überprüft, ob das ganze System den spezifizierten Anforderungen entspricht. Das System wird hier aus Sicht des Kunden oder Anwenders betrachtet, im Gegensatz zu der technischen Herangehensweise der vorangegangenen Teststufen. Das System wird in einer Testumgebung getestet, die der späteren Produktivumgebung so nahe wie möglich kommt. Daraus ergibt sich auch, dass auf den Einsatz von Testtreibern und Platzhaltern verzichtet werden soll.
[vgl. Linz2005, S. 58]

Das System wird über externe Schnittstellen getestet, also über die Benutzeroberfläche oder über Importschnittstellen.

Da bei einem Systemtest von außen getestet wird, ist hier die Unterscheidung zwischen objektorientierten und strukturierten Systemen nicht mehr so stark wie beim Integrationstest. Allerdings ist die Menge der zu bearbeitenden Testfälle unterschiedlich, da objektorientierte Systeme häufig komplexer sind und viele Abhängigkeiten haben.

Beim Systemtest werden mindestens drei Testarten unterschieden: Umgebungstest, Funktionstest und Performanz/Belastungstest.
[vgl. Sneed2002, S. 231]

Ein Softwaresystem befindet sich in zwei Umgebungen: in der technischen Systemumgebung und in der Organisationsumgebung. Es ist daher wichtig, dass bei einem Systemtest die Kompatibilität und die Interoperabilität mit beiden Umgebungen geprüft werden.

Zur Systemumgebung zählen die Hardware-Konfiguration, Basissoftware und Middleware. Bei verteilten Systemen gehören zur Hardware nicht nur alle Serverrechner, sondern auch alle Clientrechner. Das Ziel des Systemtest in Bezug auf die Systemumgebung ist also sicherzustellen, dass die Software mit allen möglichen Hard- und Software Zusammenstellungen zusammenarbeitet. Auch Performanz- und Sicherheitsaspekte spielen eine wichtige Rolle.

Für die Organisationsumgebung ist es wichtig, dass die Software sich in die Geschäftsprozesse eingliedert. Die Software sollte alle wichtigen betrieblichen Vorschriften und Benutzerkriterien erfüllen. Dabei handelt es sich unter anderem um folgende Kriterien: Datenschutz, ergonomische Normen für Benutzeroberflächen, Abbildung der Geschäftsprozesse etc.
[vgl. Sneed2002, S. 232-234]

Der Funktionstest soll überprüfen, ob die Software alle fachlichen Anforderungen ausreichend erfüllt. Dabei ist es Voraussetzung, dass alle Funktionen genau spezifiziert sind. Dies kann in einem Fachkonzept, einer Systemspezifikation oder einem Benutzerhandbuch geschehen. Die Testfälle decken dann alle spezifizierten Funktionen ab und prüfen diese

anhand des entsprechenden Dokuments. Dieses Dokument wird auch als Testorakel bezeichnet.
[vgl. Sneed2002, S. 234]

Nachdem die Klassen-, Integrations- und Funktionstests erfolgreich abgeschlossen wurden, empfiehlt es sich, gerade bei komplexen Client/Server-Systemen, aber auch bei Host-Systemen, einen Performanz- und Belastungstest durchzuführen. Unter Umständen kann man auch mit bestimmten Prototypen in einer früheren Phase Belastungstests durchführen. Das Ziel ist es, das Verhalten des Systems bei Lastspitzen und die Leistungsfähigkeit bei Belastung zu testen.

Für den Performanztest müssen viele Clients simuliert werden, die gleichzeitig auf das System zugreifen. Es können Testfälle aus dem Integrationstest verwendet werden, die auf mehreren Clients ausgeführt werden. Der Belastungstest für Client/Server-Anwendungen soll das Verhalten des Systems überprüfen, wenn ein hohes Datenaufkommen generiert wird.
[vgl. Sneed2002, S. 243]

Für den Performanztest sind automatisierte Testfälle sehr hilfreich. So können mit einigen Tools viele Clients an einem Rechner simuliert werden, die zugleich die automatisierten Tests ausführen. Eine manuelle Testausführung an vielen Clients bedeutet einen erheblichen Mehraufwand.

2.1.4 Abnahmetest

Bevor eine Software beim Kunden in Betrieb genommen werden kann, erfolgt der so genannte Abnahmetest. Der Abnahmetest kann der einzige Test sein, an dem der Kunde direkt beteiligt oder für den er selbst verantwortlich ist. Typischerweise kann unter anderen auf vertragliche Akzeptanz und Benutzerakzeptanz getestet werden.

Zwischen dem Kunden und dem Hersteller wird vor der Entwicklung ein Entwicklungsvertrag geschlossen, in dem die Abnahmekriterien festgehalten sind. Beim Abnahmetest prüft der Kunde die Einhaltung dieser Kriterien. Es ist sicherlich ratsam, dass der Entwickler zuvor in seinem Systemtest diese Kriterien bereits überprüft hat. Im Gegensatz zu dem Systemtest findet der Abnahmetest in der Systemumgebung des Kunden statt. Für die Abnahmetests können durchaus die gleichen Testfälle verwendet werden, die auch schon im Systemtest ausgeführt wurden. Aufgrund der unterschiedlichen Umgebungen kann es durchaus zu Abweichungen kommen, die beim Systemtest nicht aufgetreten sind.

Wenn Kunde und späterer Anwender nicht identisch sind, ist es ratsam, die Akzeptanz der Benutzer zu testen. Jeder Anwender hat andere Erwartungen an eine Software, und falls das System als zu „umständlich“ empfunden wird, kann es zum Scheitern des Softwareprojektes führen, selbst wenn das Programm funktional alle Anforderungen erfüllt. Für den Akzeptanztest sollten von jeder Anwendergruppe Tests durchgeführt werden, die den typischen Anwendungsszenarien entsprechen. Es kann auch sinnvoll sein, diese Tests bereits in früheren Projektphasen durchzuführen, da starke Akzeptanzprobleme möglicherweise zu einem späten Zeitpunkt gar nicht oder nur mit einem erheblichen Aufwand behoben werden können.

Anwendungen, die in vielen verschiedenen Umgebungen zum Einsatz kommen, wie es bei Standardsoftwareprojekten der Fall ist, können einem Feldtest unterzogen werden. Es wäre für den Hersteller sicherlich sehr kostenintensiv, wenn er alle möglichen Produktivsysteme selber nachstellen würde. In einem solchen Fall wird die Software, nach erfolgreichem Systemtest, den späteren Anwendern zur Verfügung gestellt. Diese Tests werden auch Alpha- bzw. Beta-Tests genannt. Wobei Alpha-Tests beim Hersteller stattfinden und Beta-Tests beim Kunden.
[vgl. Linz2005, S.61-64]

2.2 Klassifizierung von Prüftechniken

Generell lassen sich Softwaretests nach verschiedenen Kriterien klassifizieren. So gibt es die Einteilung in Black Box und White Box Verfahren.
[vgl. Linz2005, S. 105 ff.]

In [Liggesmeyer2002, S. 36] steht hingegen: „Die Unterteilung der Testtechniken in White Box- und Black Box-Techniken muss als zu grob nach dem heutigen Stand des Wissens betrachtet werden.“ Hier wird eine Unterteilung in statische und dynamische Testtechniken vorgeschlagen.

Im Folgenden werden beide Möglichkeiten der Klassifikation vorgestellt.

2.2.1 Black Box und White Box Prüftechniken

Bevor man mit dem Testen einer Software beginnen kann, sollte man sich überlegen, was genau getestet werden soll. Sicherlich ist ein Ad-hoc-Ansatz möglich, wird aber meistens nicht zu den erwünschten Ergebnissen führen. Es werden nur die Programmteile getestet, die man zufällig ausgewählt hat. Dabei besteht die Gefahr, dass wesentliche Tests vergessen werden und dass Zeit mit überflüssigen Tests vergeudet wird. Es sollten also systematisch Testfälle entworfen werden.
Für diesen Entwurf gibt es unterschiedliche Prüftechniken, die sich in Black Box- und WhiteBoxPrüftechniken unterteilen lassen. Bei WhiteBox Prüftechniken wird der Programmcode zur Erstellung der Testfälle herangezogen, bei Black Box Prüftechniken nicht. So kann es durchaus sein, dass der Quellcode einer zu testenden Software gar nicht vorliegt und auf die Black Box Prüftechniken zurückgegriffen werden muss. Dies ist zum Beispiel bei den Projekten, die in dieser Arbeit besprochen werden, der Fall.
[vgl. Linz2005, S. 208]

2.2.1.1 White Box Prüftechniken

Das Kennzeichen der White Box Prüftechniken ist, dass für die Erstellung der Testfälle die innere Struktur der Software, also der Quellcode, verwendet wird. Bei diesen Tests kann der Quellcode gegebenenfalls verändert oder erweitert werden. Das Ziel ist, dass möglichst viele Quellcodeteile zur Ausführung gebracht werden. Die zu erreichende Überdeckung mit Testfällen sollte vorher festgelegt werden.

2.2.1.2 Black Box Prüftechniken

Bei Black Box Tests werden die Testfälle anhand der Spezifikation erstellt und nicht unter Zuhilfenahme des Quellcodes. Da die Eingabe und Auswertung aller möglichen Eingabewerte aufgrund der großen Anzahl an Kombinationsmöglichkeiten nicht sinnvoll ist, muss eine sinnvolle Auswahl an Tests getroffen werden. Zum Treffen dieser Auswahl gibt es verschiedene Herangehensweisen.

2.2.2 Statische und dynamische Prüftechniken

Nach [Liggesmeyer2005] lassen sich Prüftechniken in zwei Klassen einteilen: in die statischen und die dynamischen Prüftechniken. Hauptmerkmal bei dieser Unterscheidung ist, dass bei den statischen Tests die zu testende Software nicht zur Ausführung gebracht wird, im Gegensatz zu den dynamischen Prüftechniken.

2.2.2.1 Statische Prüftechniken

Diese Prüftechniken werden auch als verifizierende oder analysierende Techniken bezeichnet. Charakteristisch hierbei ist, dass die Software nicht ausgeführt wird und dass keine Testfälle generiert werden. Für die Analysen ist prinzipiell keine Unterstützung durch Rechner erforderlich. Eine vollständige Aussage über die Korrektheit oder Zuverlässigkeit kann mit statischen Prüfungen nicht erzeugt werden.
[vgl. Liggesmeyer2002, S. 40]

Zu den statischen Prüftechniken zählt die Datenflussanalyse. Die Beurteilung der Korrektheit erfolgt aufgrund der Erkennung von Datenflussanomalien. [vgl. Linz2005, S. 88] Eine Datenflussanomalie ist beispielsweise ein Zugriff auf eine Variable, die vorher nicht initialisiert wurde. Da in der objektorientierten Programmierung sehr viel mit Datenzugriffen, zum Beispiel bei Klassen mit vielen Attributen, gearbeitet wird, ist die Datenflussanalyse als Modultest hier sehr geeignet. In der Praxis findet sie allerdings kaum Verwendung. Die Tests sind recht aufwendig und ohne entsprechende Tools kaum durchführbar, aber genau diese sind eher selten.
[vgl. Liggesmeyer2002, S. 170]

Eine weitere statische Prüftechnik, die hier kurz vorgestellt werden soll, ist die Kontrollflussanalyse. Auch diese Technik ist eher für den Modultest geeignet. Hier wird das Augenmerk auf die Kontrollstrukturen gelegt. Auftretende Kontrollflussanomalien können Sprünge aus Schleifen oder Programmstücke mit mehreren Ausgängen sein. Hierbei muss es sich nicht um Fehlerzustände handeln, sie stehen aber im Widerspruch zu den Grundsätzen der strukturierten Programmierung.
[vgl. Linz2005, S. 90]

2.2.2.2 Dynamische Prüftechniken

Die dynamischen Prüftechniken sind dadurch charakterisiert, dass die zu testende Software ausgeführt und mit konkreten Eingabewerten (Testdaten) versehen wird. Die dynamischen Prüftechniken können in der realen Betriebsumgebung zum Einsatz kommen. Allerdings

bringen sie keinen Beweis für die Korrektheit, da es sich meistens um Stichprobenverfahren handelt. Ein Test mit allen Eingabemöglichkeiten, ein erschöpfender Test, ist aufgrund der hohen Komplexität von Programmen nicht praktikabel. Trotzdem kann man mit dynamischen Prüftechniken sinnvoll prüfen.

Ob eine Funktion korrekt oder nicht korrekt ausgeführt wird, kann im Grunde nur für die verwendeten Testdaten mit Sicherheit gesagt werden. Man muss also auf die Korrektheit anderer, nicht getesteter Fälle schließen. Somit ist die Wahl der Testfälle ausgesprochen wichtig. Man sollte darauf achten, dass diese repräsentativ, fehlersensitiv, redundanzarm und ökonomisch sind. [vgl. Liggesmeyer2002, S. 36]

2.3 Regressionstests

Da nicht davon ausgegangen werden kann, dass eine Software nach ihrer Auslieferung nicht mehr weiterentwickelt werden muss, sind bestimmte Techniken zum Test von neuen Produktversionen nötig. Software wird weiterentwickelt, wenn sich die Anforderungen geändert haben oder wenn Defekte behoben werden sollen. Hierbei muss sichergestellt werden, dass durch die Änderungen erkannte Fehler behoben wurden, keine neuen Fehler entstanden sind und dass keine maskierten Fehler durch die Korrektur eines anderen Fehlers wirksam werden. Für diesen Zweck werden Regressionstests eingesetzt.
[vgl. Linz2005, S. 65ff]

Die Regressionstests fallen in die zuletzt angesprochene Kategorie der dynamischen, diversifizierenden Prüftechniken. Es werden also die Ergebnisse verschiedener Programmversionen miteinander verglichen.

Es werden Testfälle generiert, die möglichst viele Funktionalitäten, die in der Anforderung spezifiziert sind, abdecken. Diese Testfälle werden ausgeführt und die Ergebnisse anhand der Spezifikation verifiziert. Die so entstandenen Testfälle und Testergebnisse stellen die Referenztestfälle für folgende Regressionstests dar. Wird nun eine neue Version der zu testenden Software erstellt, so werden die Referenztestfälle mit dieser Version erneut abgearbeitet und die Ergebnisse mit denen der Referenztestfälle verglichen. Werden hierbei Abweichungen festgestellt, kann dies gewollt sein, eben dann, wenn ein Fehler der Vorgängerversion korrigiert wurde. Andernfalls ist in der neueren Version ein Fehler hinzugekommen, oder es wurde ein bereits bestehender Fehler demaskiert.

Bei der Durchführung von Regressionstests ist darauf zu achten, dass sich die jeweiligen Programme vor der Ausführung der Tests in dem selbem Zustand befinden, da es sonst zu falschen Fehlererkennungen kommt.

Bei der manuellen Ausführung von Regressionstests müssen also Testdaten und Testschritte, die in Testdokumentationen festgehalten sind, abgearbeitet werden. Die hierbei entstehenden Ausgaben der Software, die getestet wird, müssen anschließend mit den erwarteten Ergebnissen verglichen werden. Je mehr Testfälle abgearbeitet werden, desto eher kann es passieren, dass der Tester eine Abweichung übersieht.

Da dokumentierte Arbeitsschritte abgearbeitet werden und Ausgaben mit ebenfalls dokumentieren Werten verglichen werden, bietet sich eine Automatisierung von Regressionstests an. Ein manueller Eingriff in den Testablauf ist im Allgemeinen nur nötig,

wenn Abweichungen zwischen den Referenz- und den Regressionsergebnissen festgestellt wurden. Mit einer entsprechenden Automatisierung kann viel Zeit und somit Kosten gespart werden, und es werden keine Abweichungen übersehen. Es ergeben sich also wirtschaftliche und technische Vorteile.
[vgl. Liggesmeyer2002, S. 187-189]

2.4 Anforderungen

Das Ziel von Testen ist die Überprüfung der Anforderungen an die Software. Dies ist gilt sowohl beim manuellen als auch beim automatisierten Testen: Man möchte wissen, ob die Software die Anforderungen, die an sie gestellt werden, in einem zufriedenstellenden Maß erfüllt. Dafür ist es zuerst einmal absolut notwendig, dass diese Anforderungen ausreichend detailliert definiert und dokumentiert sind. Bei einem automatisierten Test müssen die Anforderungen noch detaillierter und sorgfältiger dokumentiert sein, da von einem Programm die Richtigkeit überprüft werden soll. Ein Tester kann bei einer ungenauen Spezifikation immer noch entscheiden, ob das Ergebnis nun richtig oder falsch ist. Wobei diese Entscheidung allerdings nicht zwangsläufig korrekt sein muss.

Beim Erfassen der Anforderungen sollte also sehr sorgfältig vorgegangen werden. Hier entstehen die meisten Fehler in einem Projekt. Diese Fehler fallen auch selten bei den Tests auf, da die Tests genau auf die Erfüllung dieser Anforderungen ausgerichtet sind. Sind die Anforderungen falsch, so sind auch vermeintlich richtige Testergebnisse falsch.

Die Anforderungen sollten also einem ausführlichen Review-Prozess unterliegen. Dabei sollten folgende Aspekte beachtet werden:

Mehrdeutigkeiten vermeiden	Da Menschen aus verschiedenen Arbeitsbereichen zusammen arbeiten, kann jeder ein anderes Verständnis für ein Problem haben.
Redundanzen vermeiden	Redundanzen erscheinen unproblematisch, allerdings besteht die Gefahr, dass bei Änderungen, z.B. während eines Reviews, nicht alle Anforderungen bearbeitet werden. Es entstehen Widersprüche.
Widersprüche	Widersprüche, die sich in eine umfangreiche Liste an Anforderungen eingeschlichen haben, sind nur sehr schwer zu identifizieren.
Ungenaue Angaben	Der Kunde hat sicher ganz genaue Vorstellungen, wie das Produkt aussehen soll. Werden diese genauen Vorstellungen nur ungenau spezifiziert, werden die Entwickler mit Sicherheit an den Kundenvorstellungen vorbei entwickeln.

[vgl. Rupp2007, S. 26]

Anforderungen lassen sich in funktionale und nicht-funktionale Anforderungen unterteilen. Entsprechende Tests werden auch funktionale oder nicht-funktionale Tests genannt, je nachdem, auf welcher Art von Anforderung sie basieren. Funktionale Anforderungen

beschreiben das Verhalten, das ein System oder ein Teil des Systems erbringen soll. Nicht-funktionale Anforderungen beschreiben, mit welcher Qualität die funktionalen Anforderungen erfüllt werden. Wichtige Kriterien für nicht-funktionale Anforderungen sind Zuverlässigkeit, Benutzbarkeit oder Effizienz.
[vgl. Linz2005, S. 69-73]

2.5 Testfallerstellung

Nachdem das Ziel, also die Anforderungen, definiert wurden, müssen Testfälle erstellt werden, die aus diesen Anforderungen resultieren. Dies ist jedenfalls das Vorgehen beim Black Box Test, bei dem, im Gegensatz zum White Box Test, keine Kenntnisse aus dem Quellcode entnommen werden. Der Black Box Test überprüft das Verhalten der Software gegenüber den Erwartungen aus den Anforderungen. Es sollten möglichst viele Anforderungen mit Testfällen verifiziert werden. Der Quotient aus mit Test abgedeckten Anforderungen und der Gesamtzahl der Anforderungen wird Überdeckungsgrad genannt. Dieser sollte möglichst hoch sein. Der Überdeckungsgrad wird bei der Ausführung der Tests auch als Testendekriterium herangezogen. Wenn also ein vorher festgelegter Überdeckungsgrad mit ausgeführten Tests erreicht wurde, kann man den Test für beendet erklären. Je nach Testergebnissen muss dann die Entscheidung gefällt werden, ob die Software überarbeitet werden muss oder ob sie die Anforderungen erfüllt.

Eine Schwierigkeit beim Erstellen der Testfälle liegt darin zu entscheiden, ab wann eine Anforderung ausreichend abgedeckt ist. Dynamische Tests sind Stichproben, und aufgrund der Vielzahl an möglichen Eingaben ist es unmöglich, alle Kombinationen zu testen. Man stelle sich nur einen einfachen Taschenrechner vor, der lediglich zwei reelle Zahlen miteinander addiert. Schon gibt es unendlich viele Möglichkeiten an Additionen und somit auch eine unendliche große Anzahl an Testfällen. Es ist offensichtlich, dass es nicht sinnvoll ist, wenn man alle diese Möglichkeiten ausprobiert, bevor man sicher sagen kann, dass dieser Taschenrechner die Addition beherrscht. Wenn er 5 + 3 richtig berechnen kann, kann davon ausgegangen werden, dass er auch bei der Eingabe von 5 + 4 zu einem richtigen Ergebnis kommt. Man spricht hier von der Bildung von Äquivalenzklassen. Mit Hilfe der Äquivalenzklassen lässt sich die Anzahl der möglichen Eingaben reduzieren. Die Problematik liegt hier in der Identifizierung der Klassen. In dem Beispiel mit dem Taschenrechner könnte man folgende gültige Äquivalenzklassen identifizieren, also solche, die eine gültige Ausgabe erzeugen sollten:

[positive Zahl; positive Zahl], [negative Zahl; negative Zahl],
[positive Zahl; negative Zahl], [negative Zahl; positive Zahl]

Bei der Erstellung der zugehörigen Testfälle sollten auch die Grenzen beachtet werden. Hier ist die Grenze die 0. In der Beispielbeschreibung wurde gesagt, dass es sich um die Addition von reellen Zahlen handelt, also können noch weitere Äquivalenzklassen erstellt werden, zum Beispiel unter Berücksichtigung von Nachkommastellen.

Neben gültigen Äquivalenzklassen gibt es auch ungültige Äquivalenzklassen. Diese enthalten eben solche Werte, die keine gültigen Eingaben darstellen. In diesem Beispiel wäre es unter anderem die Eingabe von Buchstaben. Mit ungültigen Äquivalenzklassen wird überprüft, ob die Software in der Lage ist, Fehler korrekt zu erkennen und abzufangen.

Eine gute Quelle für Testfälle sind Anwendungsfälle einer Software. Im betrieblichen Umfeld sind diese die Abbildung der Geschäftsabläufe. Die Geschäftsabläufe müssen hierfür dokumentiert sein, mit Vor-, Nachbedingung und Szenarien. Bei diesem „Funktionstests mit Anwendungsfällen“ wird die Software systematisch mit sinnvollen Anfangszuständen und Eingaben, die einen Geschäftsablauf abbilden, getestet. Mögliche Fehler, die hierbei aufgedeckt werden können, sind unvollständig implementierte Anwendugsfälle, fehlerhaft implementierte Geschäftslogik und nicht beachtete Abhängigkeiten zwischen den Anwendungsfällen.
[vgl. Sneed2002 S.238-239]

2.6 Risikoanalyse und Priorisierung von Testfällen

Bei dynamischen Tests ist ein vollständiger Test nicht möglich. Jeder Test ist nur eine Stichprobe. Doch auch die Anzahl dieser stichprobenartigen Tests kann ein sehr hohes Maß annehmen, besonders bei komplexeren Softwareprojekten. Aufgrund von Ressourcenmangel und Termindruck kann es durchaus sein, dass auch bei diesen Stichproben nicht alle Tests ausgeführt werden können. Es muss also entschieden werden, welche Tests tatsächlich ausgeführt werden. Dabei spielt auch die Reihenfolge der Testausführung eine wichtige Rolle. Sollte man während der Testausführung bemerken, dass die Zeit bis zur Deadline nicht ausreicht, um wie geplant zu testen, sollten doch die wichtigsten Tests erledigt sein. Es wäre fatal, wenn durch eine Fehlplanung oder das Eintreten von unerwarteten Ereignissen essentielle Funktionen nicht überprüft werden können. Außerdem können schwerwiegende Probleme frühzeitig erkannt werden. Eine eventuell aufwendige Korrekturarbeit ist somit weniger kritisch für das Einhalten des Projektplans. Daraus ergibt sich also die Notwendigkeit einer Priorisierung, auch wenn geplant ist, alle ermittelten Testfälle auszuführen.

Die Testfälle müssen also in wichtige und weniger wichtige unterschieden werden. Und das am besten mit verschiedenen Abstufungen, damit eine Sortierung vorgenommen werden kann. Als sinnvolles Kriterium ist das Risiko anzusehen, wobei Risiko als das Produkt aus der Höhe des möglichen Schadens und der Schadenswahrscheinlichkeit anzusehen ist. Der Begriff Schaden bezeichnet alle Kosten, die aus einer Fehlfunktion der Software resultieren können. Die Wahrscheinlichkeit hängt von der Art der Benutzung der Software ab. Dies macht eine genaue Risikobewertung schwierig.
[vgl. Linz2005, S. 184-185]

Bei Eurogate ist es üblich, den Schaden in die Kategorien A-C zu unterteilen. Dabei haben die Kategorien folgende Bedeutung:

- A: Im Fehlerfall ist die Arbeitsfähigkeit des Betriebs nicht mehr gewährleistet und Umsysteme sind betroffen.
- B: Im Fehlerfall treten Einschränkungen im täglichen Arbeitsablauf auf, die aber auf das eigentliche System beschränkt bleiben. Keine Auswirkung auf Umsysteme.
- C: Im Fehlerfall ist lediglich mit leichten Einschränkungen zu rechnen.

Die Fehlerwahrscheinlichkeit wird aus der Häufigkeit der Benutzung der jeweiligen Funktionalität ermittelt. Es erfolgt eine Einteilung von 1-3, wobei Funktionalitäten mit der Fehlerwahrscheinlichkeit 1 am häufigsten verwendet werden. Somit hat eine Funktionalität mit der Risikobewertung A-1 die höchste Priorität.

Eine geschickte Einteilung in die entsprechenden Kategorien ist keine einfache Aufgabe.

Bei Eurogate wird die Fehlerwahrscheinlichkeit im COIN-System daran festgemacht, wie häufig eine Funktion aufgerufen wird. Dafür werden im laufenden Betrieb Statistiken geführt. Es wird also davon ausgegangen, dass es wahrscheinlicher ist, dass ein Fehler im Betrieb auftritt, je häufiger eine Funktion verwendet wird. Ein Alternative wäre hier die Erstellung einer Fehlerprognose. Dies kann auf Basis verschiedener Metriken anhand des Quellcodes geschehen. Auch Analysen über die Komplexität und die Vererbungsstruktur können herangezogen werden. Ein entsprechendes Verfahren ist in [Gericke2007] beschrieben.

Eine Einteilung in die Kategorien A-C für den möglichen Schaden ist nur durch Fachleute möglich, die sich sehr genau in den entsprechenden Projekten auskennen. Der Aufwand einer solchen Analyse kann je nach Projekt sehr hoch sein. Während des Testprozesses kann möglicherweise deutlich werden, dass die Bewertung nicht korrekt war. Werden beispielsweise in als relativ unbedenklich eingestuften Bereichen übermäßig viele Fehler gefunden, kann es notwendig sein, eine Neupriorisierung vorzunehmen, die ihrerseits auch wieder mit einem hohen Aufwand verbunden sein kann.

Ein Ansatz zur Verbesserung dieser Problematik mit dem ranTEST-Ansatz (risiko- und anforderungsbasiertes Testen) ist in [Bauer2007] beschrieben. Im Groben beinhaltet dieser Ansatz die Dokumentation von Anforderungen durch Anwendungsfälle, die risikobewertet werden. Auf Basis dieser Anforderungen werden Testmodelle als Zustandsautomaten erstellt, deren Transitionen mit den entsprechenden Risiken gewichtet sind. Szenarien, die sich aus möglichen Wegen vom Start- zum Zielzustand ergeben, können anhand ihrer Risikoabdeckung eingeordnet werden. Da diese risikobasierte Ableitung und Priorisierung automatisiert werden kann, ergibt sich eine erhebliche Aufwandseinsparung. Auch eine Neupriorisierung lässt sich erheblich komfortabler durchführen. Nähere Informationen zu diesem Projekt gibt es im Internet bei der Universität Duisburg-Essen unter http://www.sse.uni-due.de/wms/de/index.php?go=204#web und auf der offiziellen Projektseite www.rantest.de.

3 Testautomatisierung

Testen und Testautomatisierung sind zwei verschiedene, auf einander basierende Handlungen. Mit dem Test wird versucht, Fehler zu finden, während das Ziel der Testautomatisierung ist, effizient und langfristig ökonomisch testen zu können. Um dies zu erreichen, werden die Testaktivitäten durch die Entwicklung und Ausführung von Testskripts unterstützt. Meistens kommen dabei Automatisierungstools zum Einsatz. In diesem Kapitel wird erläutert, welche Möglichkeiten die Automatisierung bietet und was beachtet werden sollte.

3.1 Automatisierter Vergleich

Automatisiertes Testen ist nicht nur das Aufzeichnen und Wiedergeben von Testabläufen, ein wichtiger Bestandteil ist auch der automatisierte Vergleich der Testergebnisse mit den Sollwerten. Werden Tests manuell ausgeführt, ist es häufig der Fall, dass der Tester sich das Ergebnis des Tests anschaut und dann entscheidet, ob es richtig oder falsch ist. In diesem Fall ist das erwartete Ergebnis im Kopf des Testers vorhanden. In der Testdokumentation steht eventuell: „Überprüfe, ob die Software das Richtige macht“. Das kann ein Tester sicherlich erfüllen, sofern er das entsprechende Hintergrundwissen hat, auch wenn es sich dabei nicht um ein empfehlenswertes Vorgehen handelt. Bei der Automatisierung ist dies nicht möglich. Um automatisiert überprüfen zu können, ob sich die Software korrekt verhält, muss man sich präzise Gedanken darüber machen, wie diese Validierung auszusehen hat. Es besteht die Möglichkeit genau vorherzusagen, wie die Software sich zu verhalten hat, oder alternativ die Ergebnisse eines Testdurchlaufs zu speichern, manuell zu überprüfen und für weitere Tests als Referenzdaten zu benutzen.

Ob es besser ist, die Ergebnisse vorherzusagen, oder ob man Referenzergebnisse zur Validierung verwendet, hängt von verschiedenen Faktoren ab:

1. Die Menge der Ergebnisse: Wenn eine einzelne Zahl oder ein kurzer String überprüft werden soll, ist es einfach, das Ergebnis vorherzusagen. Ist das Ergebnis aber ein Bericht, der mehrere Seiten lang ist, kann es erheblich einfacher sein, mit Referenzdaten zu arbeiten.

2. Kann ein Ergebnis überhaupt vorhergesagt werden? Wird mit Echtdaten gearbeitet, kann es eventuell sein, dass man keine Vorhersagen machen kann.

3. Verfügbarkeit der AUT (Application under test): Man kann die ersten Referenzergebnisse erst erstellen, wenn man die Software zur Verfügung hat. Aber aus Zeitgründen möchte man die Tests schon vorbereiten, bevor man die eigentliche Software erhält.

4. Qualität der Überprüfung: Meistens ist es besser, die Ergebnisse vorherzusagen, als die ersten Testergebnisse manuell zu überprüfen. So wird die Gefahr umgangen, dass bei der manuellen Prüfung Fehler übersehen werden und alle weiteren automatischen Testausführungen gegen falsche Referenzwerte geprüft werden.

Das automatisierte Vergleichen bringt die meisten Vorteile im Bereich der Testautomatisierung. Bei der Überprüfung ist es häufig notwendig, dass sehr viele Daten, Bildschirminhalte und Zahlen verglichen werden müssen. Dies ist eine langweilige und anstrengende Arbeit. Die Wahrscheinlichkeit, dass beim manuellen Prüfen Fehler übersehen werden, ist sehr hoch. Deshalb ist es sinnvoll, diese Arbeit dem Computer zu überlassen.
[vgl. Fewster1999 S.101 ff]

3.2 Automatisierte Vor- und Nachbereitung

Unter Testvorbereitung versteht man das Schaffen von Vorraussetzungen, damit ein Test ausgeführt werden kann. Dies kann zum Beispiel das Anlegen bestimmter Datensätze sein, mit denen im Test gearbeitet werden soll. Unter Umständen ist es nötig, diese Datensätze bei jedem Testdurchlauf erneut anzulegen.

Testnachbereitung bezeichnet alle Schritte, die nach dem eigentlichen Test erforderlich sein können. Eventuell liegen die Testergebnisse an verschiedenen Stellen im System und müssen in das dafür vorgesehene Verzeichnis gelegt werden. Manche Protokolle, die keine Fehlermeldungen enthalten, können gelöscht werden, oder bestimmte Datensätze, die nur für den Test benötigt wurden, müssen wieder aus der Datenbank entfernt werden.

Es gibt eine Menge an Vor- und Nachbereitungsaufgaben. Diese Aufgaben sind normalerweise sehr zeitintensiv und müssen nach jedem Testfall ausgeführt werden. Da es sich häufig um gleiche oder ähnliche Aufgaben handelt, die nicht sehr komplex sind, bietet sich eine Automatisierung an. Meistens handelt es sich hierbei um Aufgaben wie das Kopieren von Dateien. Es ist schwer vorstellbar, von einem automatisierten Testablauf zu reden, wenn vor und nach jedem Test der Tester manuell sehr viele Dateien kopieren oder löschen muss.

Die Aufgaben der Vor- und Nachbereitung können häufig in einem Test Execution Tool genau so mit Skripten automatisiert werden wie die Tests selbst. Wenn man in diesem Tool mit Funktionen arbeiten kann, bietet es sich an, die Vor- und Nachbereitung in entsprechende Funktionen zu kapseln und an den entscheidenden Stellen in das Automatisierungsskript zu implementieren. Es sollte noch beachtet werden, dass bei der Nachbereitung andere Aufgaben anfallen können, wenn der Test nicht erfolgreich abgeschlossen wurde. In diesem Fall ist es durchaus sinnvoll, keine Bereinigung vorzunehmen, damit am Ende alle Daten zur Analyse des Fehlers vorliegen.
[vgl. Fewster1999 S.176 ff]

3.3 Erstellung von wartbaren Tests

Die Wartbarkeit bei automatisierten Tests muss einen höheren Stellenwert erhalten, als es bei manuellen Tests der Fall ist. Jedes Mal, wenn die Software angepasst wird, müssen die Tests überprüft und angepasst werden. Wird bei einer Adressbuch Anwendung ein Feld „Telefonnummer“ hinzugefügt, so muss bei einem automatisierten Testskript das Füllen dieses Feldes hinzugefügt werden. Bei einem manuellen Test steht vielleicht die Anweisung „Alle Felder sollen mit Daten gefüllt werden“. In diesem Fall wird der Tester das Telefonnummer-Feld füllen, obwohl die Testbeschreibung nicht angepasst wurde.

Es ist also entscheidend, dass die Übersicht über die Testfälle nicht verloren geht. Ein wichtiges Hilfsmittel hierzu ist die Verwendung einer Testsuite wie des HP Quality Center, das weiter unten beschrieben wird. Ein solches Tool verwaltet alle Tests und die dazugehörigen Daten zentral. Aber auch wenn man ein solches Tool verwendet, sollte beachtet werden, dass die Anzahl der Tests nicht überhand nimmt. Dieses Problem kann zum Beispiel auftreten, wenn viele Mitarbeiter Tests anlegen. Es können Redundanzen entstehen durch Testfälle, die mal als Versuch angelegt und später nicht wieder entfernt wurden. Je größer die Anzahl der Tests, desto mehr Wartungsaufwand entsteht, wenn der zu testenden Software neue Funktionalitäten hinzugefügt werden. Es ist also unbedingt darauf zu achten, dass nur wirklich sinnvolle Tests angelegt werden und in regelmäßigen Abständen der Bestand nach nicht verwendeten Tests abgesucht wird. Letzteres kann zum Beispiel dann geschehen, wenn die Tests gerade einer Wartung unterzogen werden, weil ein neues Release ansteht.

Es sollten ebenfalls darauf geachtet werden, dass die Menge der in der Testsuite gespeicherten Daten nicht zu groß wird. Auch hier tritt ein Problem mit der Wartbarkeit auf, aber es kommen noch Performanceprobleme hinzu, da diese Daten bei der Testausführung aus der Datenbank auf die Testrechner kopiert werden müssen. Häufig lassen sich die Datenbestände verringern, wenn bei der Planung mehr Zeit darauf verwendet wird zu identifizieren, welche Daten tatsächlich benötigt werden. Die Daten sollten nach Möglichkeit in einem flexiblen Format gespeichert werden. Es ist nicht immer ratsam, alle Testdaten so zu speichern, dass nur die AUT sie lesen kann. Eventuell wird eine Änderung vorgenommen, die genau dieses Datenformat betrifft. Es entsteht ein viel geringerer Wartungsaufwand, wenn die Testdaten in einem flexiblen Format vorliegen.

Die einzelnen Testfälle sollten so kurz wie möglich gehalten werden. Oft ist man versucht, lange Tests zu konstruieren, da sie automatisch ablaufen und der Computer sich daran nicht stört. Aber automatisierte Tests sind Software und müssen genau wie andere Programme getestet werden. Wurde ein langer Test fehlerhaft implementiert und führt dies bei seiner Ausführung zu einem Fehler, ist der Aufwand, diesen Fehler zu beheben, wesentlich größer als bei kleineren Testabläufen. Solche langen Tests können auch durch Abhängigkeiten zwischen mehreren Tests entstehen. Dies ist der Fall, wenn das Ergebnis des einen Tests die Testsdaten für den nächsten Test darstellen. So eine Konstellation lässt sich nicht immer vermeiden und ist häufig auch sehr effektiv. Es sollte trotzdem vorsichtig mit dieser Technik umgegangen werden. Weitere Regeln aus der Softwareentwicklung wie eine ausreichende Dokumentation und die Verwendung von Namenskonventionen sollten bei der Erstellung von automatisierten Tests ebenfalls eingehalten werden.
[vgl. Fewster1999 S.191 ff]

3.4 Metriken

Wenn eine Firma in die Verbesserung ihres Softwaretest-Prozesses oder in die Automatisierung von Tests investiert, möchte sie wissen, ob sich diese Investition gelohnt hat. Eine wichtige Metrik kann das ROI (Return on investment) sein. Folgende Tabelle zeigt eine Beispielrechnung, wie das ROI für einen verbesserten Testprozess ohne Automatisierung ermittelt werden kann.

	aktueller Prozess	verbesserter Prozess
Testkosten	10.000 €	20.000 €
Defect Detection Percentage (DDP)	70%	90%
gefundene Defects	700	900
Kosten der Defectbehebung im Test	70.000 €	90.000 €
gefundene Defects nach Release	300	100
Kosten der Defectbehebung nach Release	300.000 €	100.000 €
Summe Kosten	380.000 €	210.000 €
Ersparnis durch Prozessverbesserung		170.000 €
Investition zur Prozessverbesserung		10.000 €
ROI (Einsparung / Investition)		17x (1700%)

[vgl. Fewster1999 S. 204]

Der aktuelle Prozess kostet 10.000 € im Jahr und deckt 70% der vorhandenen Fehler auf (DDP = Defect Detection Percentage). In dieser Beispielrechnung sind 1000 Fehler in der Software enthalten. Die Kosten, diese Fehler zu beheben, liegen in der Testphase bei 100 € und nach dem Release bei 1000 €. Das Verhältnis der Kosten zwischen der Testphase und der Nach-Release Phase ist als durchaus realistisch anzusehen. Der Wert für das ROI variiert nach Anzahl der Fehler und der DDP.

Eine entsprechende Beispielrechnung für die Einführung eines automatisierten Testprozesses kann folgendermaßen aussehen:

	manueller Test	automatischer Test
Kosten für die Testfallerstellung	6.000 €	6.000 €
Toolkosten		5.000 €
Kosten Implementierung autom. Tests		11.000 €
Gesamtkosten Automatisierung		16.000 €
Kosten für einen Testzyklus	5.000 €	1.000 €
Anzahl der Zyklen pro Release	3	3
Testkosten pro Release	21.000 €	9.000 €
Einsparungen pro Release		12.000 €
Releases pro Jahr	4	4
Einsparungen pro Jahr		48.000 €
Gewinn (Einsparungen–Investition) pro Jahr		32.000 €
ROI (Einsparung / Investition)		200%

[vgl. Fewster1999 S. 205]

Es wird deutlich, dass die Einführung der Automatisierung zunächst Kosten verursacht. Diese Kosten bestehen sowohl aus Softwarekosten für die Tools als auch aus Implementierungskosten für die automatisierten Tests. Andererseits sind die Kosten pro Testzyklus sehr viel niedriger. Die Ausführung der Tests erfolgt schneller und verursacht weniger Personalkosten. Bei dieser Tabelle handelt es sich um ein einfaches Rechenmodell, bei dem einige Aspekte nicht berücksichtigt wurden. Es kann durchaus sein, dass die Kosten bei den ersten automatisierten Zyklen höher liegen, da eventuell noch Fehler in der Implementierung der Testfälle auftreten. Es kann auch sein, dass die Automatisierung keinen direkten finanziellen Vorteil erzielt, dafür den Test schneller abschließt und somit ein kürzeres Time to Market erreicht wird.

Eine Metrik, die Anzahl der gefundenen Fehler, wird bei der Automatisierung nicht betrachtet. Manuelle Tests haben in der Praxis die höhere Chance, Fehler zu entdecken. Bei der Automatisierung werden diese Fehler während der Erstellung und der ersten manuellen Ausführung der Tests entdeckt. Das bedeutet aber nicht, dass sich eine Automatisierung nicht lohnt. Die Vorteile entstehen unter anderem bei der wiederholten Ausführung von Tests, zum Beispiel bei Regressionstests.
[vgl. Fewster1999 S.203-207]

Es ist ratsam, mit Metriken den Testprozess zu überwachen. Nur so kann festgestellt werden, ob neue Techniken effektiver sind als bisherige oder ob eine weitere Optimierung dringend notwendig ist. Welche Metriken erfasst werden sollten, hängt von den Zielen ab, die verfolgt werden. Mögliche Ziele, die mit einer Testautomatisierung erreicht werden können, sind:

- konsistente, wiederholbare Tests
- Tests unbeaufsichtigt ausführen
- Fehler durch Regressionstests finden
- Tests häufiger ausführen
- höhere Softwarequalität
- gründlicher Testen
- Vertrauen in die Qualität der Software erhöhen
- Kosten reduzieren
- mehr Fehler finden
- Testphase schneller abschließen
- bessere Testdokumentation

[vgl. Fewster1999 S. 210]

Neben den anfangs beschriebenen ROI, können folgende Metriken einen Analysewert für den Testprozess darstellen:

Der Defect Detection Percentage (DDP) gibt an, wie hoch der Anteil der gefundenen Fehler gegenüber den vorhandenen Fehlern ist. Natürlich kann keine garantiert richtige Angabe dazu gemacht werden, da die Anzahl der vorhandenen Fehler nicht bekannt ist. Als Wert für die vorhandenen Fehler kann die Anzahl der Fehler herangezogen werden, die nach dem Softwaretest aufgefallen sind. Mit dem DDP lassen sich gravierende Mängel im Testprozess aufdecken oder Vergleiche anstellen, wenn der Testprozess gerade optimiert wird.

Die durchschnittliche Zeit, die benötigt wird, um einen Testfall zu automatisieren, ist ein wichtiger Anhaltspunkt, um den Aufwand für die weitere Automatisierung abzuschätzen. Werden Optimierungen am Automatisierungsprozess vorgenommen, kann mit dieser Metrik feststellt werden, ob und wie viel Zeit eingespart wird.

Es ist ebenfalls sinnvoll, die Zeit zu betrachten, die benötigt wird, um vorhandene Testfälle zu warten. Wenn die Anpassung der vorhandenen Tests für eine Aktualisierung der zu testenden Software einen zu hohen Aufwand bedeutet, ist es unter Umständen nötig, den Automatisierungsstandard zu überarbeiten.

Um einen Vorteil der Testautomatisierung zu messen, kann die Anzahl der automatisch ausgeführten Tests mit der Anzahl der Tests, die manuell hätten ausgeführt werden können, gegenübergestellt werden. Bei einer vollständigen Automatisierung ist es leicht möglich, 100% der Testfälle auszuführen, auch wenn mehrere Testzyklen durchlaufen werden. Bei

einer manuellen Testausführung wären in der gleichen Zeit nur ein Teil der Tests (vielleicht 10%) ausgeführt worden.
[vgl. Fewster1999 S.226-227]

3.5 Was kann und sollte automatisiert werden?

Grundsätzlich gilt, dass nicht zu viel auf einmal automatisiert werden sollte. Es ist sinnvoll, erst einen kleinen Anteil der Tests zu automatisieren und sich davon zu überzeugen, dass die Implementierung in der bestmöglichen Art und Weise geschehen ist. Sollte die Praxis zeigen, dass eine andere Herangehensweise notwendig ist, müssen nicht zu viele Tests aktualisiert oder neu implementiert werden. Hieraus resultiert das Problem, sich entscheiden zu müssen, mit welchen Testfällen begonnen werden sollte. Für gewöhnlich existieren Tests, die automatisiert werden können, und solche, die nicht automatisiert werden können. Im Falle der automatisierbaren Tests ist es meistens unsinnig, die zu automatisieren, bei denen der Automatisierungsprozess mehr Zeit in Anspruch nimmt als die manuelle Ausführung. Dazu ein Beispiel: Für einen Test, der manuell in 10 Minuten ausgeführt wird und einmal im Monat läuft, benötigt man 120 Minuten im Jahr. Benötigt die Automatisierung aber 10 Stunden, so hat sich diese erst nach 5 Jahren rentiert. Es können trotzdem Faktoren dafür sprechen, die Automatisierung durchzuführen. So kann es sehr kompliziert sein, den Test manuell abzuarbeiten. Vielleicht brauchen die Tester vier Versuche, um die korrekten Eingaben vorzunehmen. Dies kann zu einer hohen Frustration bei den Testern führen, und eine Automatisierung würde somit das Engagement des Testteams steigern.

Für den Test einer Applikation existieren verschiedene Arten von Tests, von denen einige leichter zu automatisieren sind als andere. Bei Funktionstests wird analysiert, ob die Software sich bei bestimmten Eingaben korrekt verhält. Es gibt also fest definierte Eingaben, Zustände und Ergebnisse. Diese Art von Tests lassen sich in der Regel gut automatisieren. Dies gilt ebenfalls für Performance- und Lasttests, die zu den nicht-funktionalen Tests gehören. Die Reaktionszeit der Software soll unter verschiedenen Bedingungen gestestet werden. Es werden also gleiche Tests sehr häufig ausgeführt oder sogar von vielen Benutzern gleichzeitig. Eine manuelle Ausführung ist hier fast unmöglich. Es ist zum Beispiel sehr schwierig, 200 Benutzer zu koordinieren, sofern man diese überhaupt zur Verfügung hat, um ein System zu testen. Auch muss man hierfür gegebenenfalls die Hardwarekosten mit berücksichtigen.

Schwer zu automatisieren sind hingegen viele andere Tests von nicht funktionalen Anforderungen. Das sind Tests, die folgende Aspekte verifizieren sollen: Wartbarkeit, Portierbarkeit, Testbarkeit, Benutzbarkeit. Es ist für ein Automatisierungstool unmöglich festzustellen, wie komfortabel eine Oberfläche ist oder ob die Farben der Bildschirmanzeige harmonieren. Für diese Fälle ist immer ein manueller Test erforderlich.

Steht man nun vor der Aufgabe zu entscheiden, welche Testfälle als erstes automatisiert werden sollen, können verschiedene Faktoren entscheidend sein. Welche wirklich relevant sind, hängt von den spezifischen Gegebenheiten ab. Man kann sich entscheiden zuerst

- die wichtigsten Tests (sofern diese zu identifizieren sind),
- die Tests der wichtigsten Funktionen,
- die Tests, die am einfachsten zu automatisieren sind,
- die Tests, bei denen sich eine Automatisierung am schnellsten bezahlt macht,
- die Tests, die am häufigsten ausgeführt werden,

- oder einen Satz an Stichprobentests, die über das System verteilt sind

zu automatisieren.

4 Projektbeschreibungen

Es folgt eine Beschreibung der Projekte, für die eine Testautomatisierung implementiert werden soll. Diese Beschreibungen stellen die Grundlage für das weitere Vorgehen dar.

4.1 COIN

Im Zuge einer 1:1 Migration soll das bisherige COIN-System durch ein System, das auf einer aktuellen Architektur basiert, ersetzt werden. Im Folgenden werden die Systemarchitektur und die Aufgaben, die mit COIN erledigt werden, beschrieben. Anschließend folgt die Beschreibung des Projektes sowie der Testanforderungen.

4.1.1 Technische Beschreibung

COIN ist eine COBOL-Entwicklung, die im Jahre 1983 in Betrieb genommen wurde. Seit dieser Zeit wurde es kontinuierlich weiterentwickelt und an die sich verändernden Anforderungen angepasst. Es läuft auf einer Unisys Host Systemumgebung mit dem Betriebssystem OS2200 und wird auf Windowsrechner mit Hilfe der Terminalsoftware Exceed von Hummingbird (www.hummingbird.com) verwendet. Die Datenhaltung erfolgt auf dem Datenbankmanagementsystem DMS in einer hierarchischen Datenbank. Im Gegensatz zu relationalen Datenbanken werden die Daten hier in einer Baumstruktur verwaltet.

Schnittstellen zu anderen System existieren nur durch den Austausch sequentieller Dateien, zum Beispiel als EDI-Nachrichten. Die Abarbeitung der Nachrichten erfolgt mit verschiedenen Batchläufen.

4.1.2 Funktionalität von COIN

COIN ist die Abkürzung für „Container Information“ und bezeichnet das administrative Container-Daten-Verwaltungsprogramm bei Eurogate. In COIN können Daten über Container erfasst, geändert und angezeigt werden. Das Erfassen der Daten kann manuell geschehen oder durch den Import von EDI-Nachrichten. Mögliche Nachrichten sind COPRAR (Container discharge/loading order message) oder BAPLIE (Bayplan/Stowage Plan Occupied And Empty Locations Message). Die COPRAR enthält Informationen darüber, welche Container auf ein Schiff geladen oder welche gelöscht werden sollen. Die BAPLIE ist der aktuelle Ladeplan eines Schiffes. Ihre Informationen beschreiben, welche Container sich an welchen Ort auf dem Schiff befinden. Auch diese Informationen lassen sich in COIN nachträglich editieren.

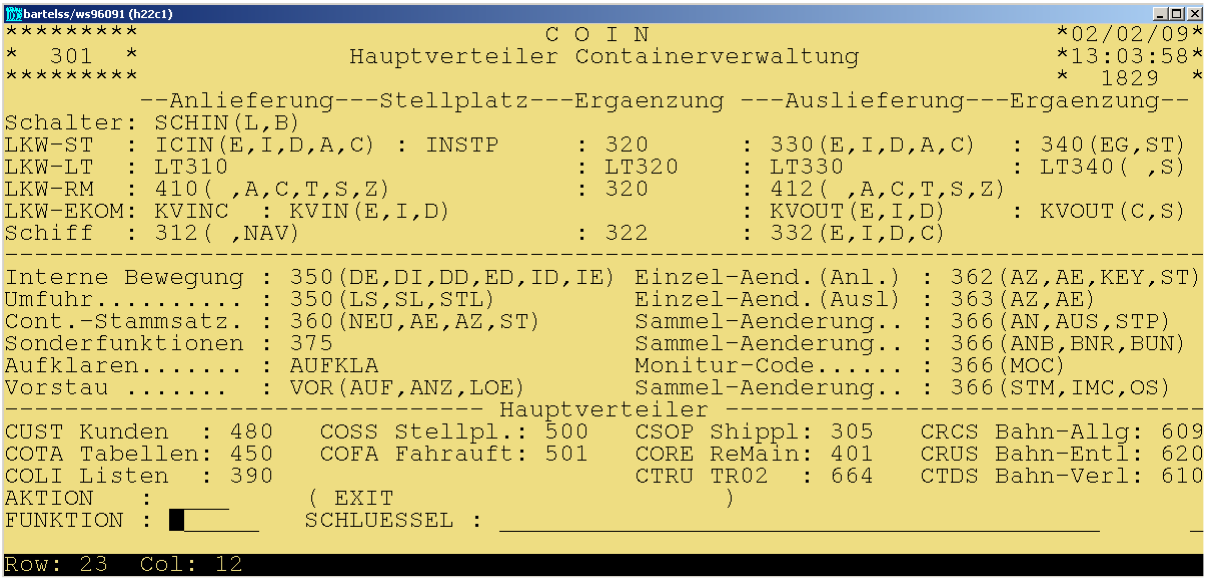

bartelss/ws96091 (h22c1)
C O I N
02/02/09
Hauptverteiler Containerverwaltung
13:03:58
* 1829 *
--Anlieferung---Stellplatz---Ergaenzung ---Auslieferung---Ergaenzung--
Schalter: SCHIN(L,B)
LKW-ST : ICIN(E,I,D,A,C) : INSTP : 320 : 330(E,I,D,A,C) : 340(EG,ST)
LKW-LT : LT310 : LT320 : LT330 : LT340(,S)
LKW-RM : 410(,A,C,T,S,Z) : 320 : 412(,A,C,T,S,Z)
LKW-EKOM: KVINC : KVIN(E,I,D) : KVOUT(E,I,D) : KVOUT(C,S)
Schiff : 312(,NAV) : 322 : 332(E,I,D,C)
Interne Bewegung : 350(DE,DI,DD,ED,ID,IE) Einzel-Aend.(Anl.) : 362(AZ,AE,KEY,ST)
Umfuhr..... : 350(LS,SL,STL) Einzel-Aend.(Ausl) : 363(AZ,AE)
Cont.-Stammsatz. : 360(NEU,AE,AZ,ST) Sammel-Aenderung.. : 366(AN,AUS,STP)
Sonderfunktionen : 375 Sammel-Aenderung.. : 366(ANB,BNR,BUN)
Aufklaren..... : AUFKLA Monitur-Code..... : 366(MOC)
Vorstau ..... : VOR(AUF,ANZ,LOE) Sammel-Aenderung.. : 366(STM,IMC,OS)
Hauptverteiler
CUST Kunden : 480 COSS Stellpl.: 500 CSOP Shippl: 305 CRCS Bahn-Allg: 609
COTA Tabellen: 450 COFA Fahrauft: 501 CORE ReMain: 401 CRUS Bahn-Entl: 620
COLI Listen : 390 CTRU TR02 : 664 CTDS Bahn-Verl: 610
AKTION : (EXIT)
FUNKTION : SCHLUESSEL :
Row: 23 Col: 12

Abbildung 4-1: COIN-Hauptmenü

Außerdem können in COIN Einzelheiten über Anlieferungen und Auslieferungen angezeigt und bearbeitet werden. Auch gibt das System Auskunft darüber, welche Container sich im Zulauf befinden. Hinzu kommt die Abwicklung der landseitigen- und seeseitigen Annahme und Ausgabe von Containern. Bei der seeseitigen Ausgabe von Containern, also beim Laden eines Schiffes, sind die Zollbestimmungen zu beachten. Auch hier gibt es ein EDI-basiertes Kommunikationssystem zwischen dem Zoll, dem Reeder, dem Terminalbetreiber, sowie weiteren beteiligten Parteien. Dieses System trägt im Hamburger Hafen den Namen „ZAPP“ (Zoll-Ausfuhr im Paperless Port). Über diese Kommunikation können Container durch die Vergabe einer B-Nr. (Bearbeitungsnummer Zoll) zum Export freigegeben oder vom Zoll gesperrt werden. Gesperrte Container dürfen nicht verladen werden, da sie entweder noch vom Zoll gesondert kontrolliert werden oder bei einer Kontrolle die Ausfuhr nicht erlaubt wurde. Diese Informationen sind ebenfalls in COIN ersichtlich.

Die Bedienung basiert auf der Eingabe von Funktionsnamen, die meistens mit einem Parameter versehen werden. Die meisten Funktionsnamen setzen sich aus der Maskennummer und der gewünschten Funktion zusammen. So bringt der Aufruf „362AZ“, gefolgt von einer Containernummer, alle Informationen über diesen Container auf den Bildschirm, während mit „362AE“ eine Maske aufgerufen wird, in der die Informationen zu diesem Container geändert werden können. Nicht alle Funktionen folgen diesem Schema. Dies betrifft sowohl die Funktionsnamen als auch die Parameter. Es gibt Funktionen, die zwei Parameter benötigen. Da aber nur ein Eingabefeld für Parameter vorhanden ist, werden beide Werte in das Feld eingetragen und mit einem Trennzeichen versehen. Einige Funktionen können auch unterschiedliche Parametertypen annehmen. Da der Parametertyp nicht immer anhand seines Wertes identifizierbar ist, wird der Wert mit einem Präfix versehen. Eine Nummer stellt zum Beispiel Typ 1 dar, während eine Nummer mit einem vorangestellten „@“ den Parametertyp 2 kennzeichnet.

4.1.3 Projektbeschreibung

Es wird beabsichtigt, sich von der Unisys-Systemumgebung zu trennen und auf eine Client/Server Umgebung mit Windows-Clients und Unix-Servern umzusteigen. Das Host-

System soll aufgrund der hohen Wartungskosten und dem bald endenden Support komplett abgelöst werden. Die Datenhaltung soll in Zukunft auf Oracle-Servern erfolgen.

Eine Neuentwicklung des Systems erscheint aufgrund der hohen Komplexität der Geschäftsabläufe als schwierig und langwierig. Deshalb soll eine weitgehend automatisierte Umsetzung des Quellcodes auf Micro Focus COBOL erfolgen. Die Konvertierung der Datenbank nach Oracle soll ebenfalls in einem hochautomatisierten Prozess erfolgen. Mit der Umsetzung wurde ein externer Dienstleister beauftragt, der bereits Erfahrung mit derartigen Projekten besitzt und eine entsprechende Auswahl an Automatisierungstools entwickelt hat. Die Datenschnittstellen zum operativen System TOPX „getHost“ und „putHost“, die bisher als COBOL-Batchprogramme realisiert sind, sollen durch Perl-Skripte ersetzt werden.

Bevor diese neue Version von COIN in Betrieb genommen wird, ist ein umfassender Test erforderlich. Dieser Test soll sowohl die Funktionalität als auch die Originaltreue der konvertierten COBOL-Anwendung und der durch Perl-Skript ersetzten Batchläufe zeigen. Außerdem soll die Integrität und die Originaltreue der Daten überprüft werden.

COIN umfasst ca. 500 verschiedene Funktionen, die in 163 Dialogen (24x80 Zeichen) aufgerufen werden können. Hinzu kommen noch ca.150 Batchläufe.

4.1.4 Testanforderung

Dieser Test hat eine sehr besondere Ausgangsbedingung. Es soll zwei Programme geben, die sich gleich verhalten und auch das gleiche Layout haben. Ein Fehler ist also eine Differenz zwischen den beiden Systemen, selbst dann, wenn ein fehlerhaftes Verhalten im Altsystem vorliegt, das im neuen System nicht vorhanden ist. Ein Test von zwei Programmen mit der gleichen Spezifikation wird Back-to-Back Test genannt.

Für die 163 Dialoge soll im Test sichergestellt werden, dass sie in folgenden Punkten mit dem COIN-Altsystem übereinstimmen:

a. Maskeninhaltlich
b. Layout
c. Tabulator-Reihenfolge der Eingabefelder
d. Programmierte Aktionen (STOP-NEXT-BACK)

Mit den Aktionen kann in Listen weiter geblättert werden (NEXT, BACK) oder es können Funktionen abgebrochen werden. Wird mit einer bestimmten Funktion das Anlegen eines Datensatzes initiiert, können in der Maske die entsprechenden Daten eingetragen werden. Erst wenn alle wichtigen Felder mit korrekten Daten gefüllt wurden, wird dieser Datensatz in der Datenbank angelegt. Ohne den Datensatz anzulegen, kann die Maske nur mit den Aktionen STOP oder EXIT verlassen werden.

Im manuellen Test würden diese Anforderungen so umgesetzt werden, dass der Tester jeden Dialog sowohl im Alt- als auch im Neusystem öffnet und Zeichen für Zeichen auf Gleichheit überprüft, anschließend die Tab-Reihenfolge kontrolliert und die Aktionen testet. Zum einen würde hier ein erheblicher Testaufwand entstehen, zum anderen ist die Wahrscheinlichkeit, dass ein Fehler übersehen wird, recht hoch. Aufgrund der automatischen Übersetzung des Neusystems ist davon auszugehen, dass nicht viele Fehler entstehen werden. Da das

Vergleichen von Bildschirmmasken als eher langweilige Tätigkeit zu betrachten ist, muss man mit einem hohen Ermüdungsfaktor rechnen. Hat der Tester bereits einige Masken erfolgreich getestet, wird seine Sorgfalt in den weiteren Masken nachlassen. Dies ist ein ganz normales menschliches Verhalten.

Weiter gefordert ist der Test der ca. 500 verschiedenen Funktionen. Aufgrund dieser großen Menge an Funktionen soll von den Fachabteilungen eine Risikobewertung erfolgen. Es ist von Eurogate gefordert, dass die Funktionen nach Benutzungshäufigkeit (1 – 3) und Kritikalität (A – C) für den Betrieb bewertet werden. Auf Basis dieser Einteilung soll eine Priorisierung der Testfälle erfolgen. Die Funktionen sollen mit verschiedenen Datensätzen ausgeführt werden. Es wird also mehrere Testfälle pro Funktion geben. Da für die Tests Daten aus dem Produktivsystem verwendet werden, müssen zu den jeweiligen Datenständen passende Datensätze festgelegt werden.

Der Test der Funktionen kann aus zwei Blickwinkeln betrachtet werden. Möchte man die Funktionalität testen, müssen Geschäftsabläufe überprüft werden, wie zum Beispiel das Anlegen eines Containers, das Aufrufen des Datensatzes, um zu prüfen, ob er tatsächlich angelegt wurde. Anschließend könnte man die Daten des Containers verändern, wieder prüfen, ob die Änderungen in die Datenbank übernommen wurden, und anschließend den Datensatz wieder löschen. Ein solcher Geschäftsablauf umfasst bereits den Aufruf von unterschiedlichen Funktionen.

Vorher erscheint es allerdings sinnvoll, überhaupt zu prüfen, ob alle Funktionen in das neue System übernommen wurden. Aufgrund des risikobasierten Testens würden einige Funktionen gar nicht aufgerufen werden. Diese Funktionen sind dann zwar sicherlich als wenig benutzt und unkritisch eingestuft, aber es könnte trotzdem fatale Folgen haben, wenn ihr Fehlen erst nach der Testphase festgestellt wird.

Die genaue Vorgehensweise für den Test der Datenbank und der Batchläufe ist noch nicht spezifiziert worden. Deswegen werden diese in der folgenden Betrachtung nicht oder nur am Rande betrachtet.

4.2 TOPX

Dieses Kapitel behandelt ein zweites Softwareprojekt, welches in Hinblick auf eine Unterstützung durch automatisierte Tests betrachtet wird. Im Gegensatz zum textbasierten COIN Programm handelt es sich bei TOPX um eine Anwendung mit einer grafischen Benutzeroberfläche, die auf dem GUI-Tool QT von Trolltech (weitere Informationen unter: http://qtsoftware.com/) basiert.

4.2.1 Technische Beschreibung

TOPX läuft auf einem Sun Solaris 10 Applikationsserver, für die Datenhaltung wird ein Oracle Datenbankserver verwendet. TOPX wird von Windows Workstation mit der Host-Emulation Hummingbird Exceed ausgeführt. Die Anzeige der GUI erfolg mittels des X Window Systems in der Version 11. Hierbei agiert die Windows Workstation als X Server, sie stellt also das Display für die Applikation bereit.
[vgl. O'Reilly2008]

Via Handheld-Terminals (HHT) können Daten mit TOPX ausgetauscht werden. Diese HHT's kommen unter anderen in den Van Carriern (VC) zum Einsatz. Auf diesen Terminals werden die Start- und Zielpositionen von Containern angezeigt, die von dem Van Carrier bewegt werden sollen. Im Gegenzug werden die neuen Positionen nach einer Umfuhr (Bewegen eines Containers an einen anderen Stellplatz) des Containers an TOPX bestätigt. Diese Fahraufträge, die von den VCs abgearbeitet werden, werden von TOPX generiert. Die Kommunikation mit dem COIN System folgt über die Schnittstellen „Gethost" und „Puthost". Deren Aufgabe ist es, Daten zwischen dem TOPX Oracle-Server und dem COIN DMS Datenbankserver auszutauschen.

4.2.2 Aufgaben von TOPX

TOPX ist ein grafisches Planungstool zur zentralen Administration der operativen Abläufe auf dem Containerterminal Hamburg. Es ist eine proprietäre Software, die vom Hersteller an die spezifischen Anforderung der Kunden angepasst wird. Das System bietet Unterstützung und Optimierung in der Platzplanung. Die Platzplanung beschreibt, welcher Container zu welcher Zeit an welchem Platz auf dem Terminal stehen soll. Für gewöhnlich wird ein Container, der für den Export bestimmt ist, per LKW oder Bahn angeliefert. Die Containerdaten sind in COIN erfasst, und dem Container wird von TOPX ein Stellplatz zugewiesen. In der Zeit, bevor ein Schiff am Kai anlegt, werden alle Container für dieses Schiff in die Nähe der Containerbrücke gefahren. Auch diese Planungsaufgabe wird mit TOPX erledigt. Die Container werden in Reihen aufgestellt und bis zu 3 Container hoch gestapelt. Nach Möglichkeit stehen alle Container für dasselbe Schiff mit demselben Zielhafen in einer Reihe. Ziel der Optimierung ist es, dass die Van Carrier möglichst kurze Wege zurück legen müssen und dass keine Wartezeiten beim Löschen und Beladen des Schiffes entstehen.

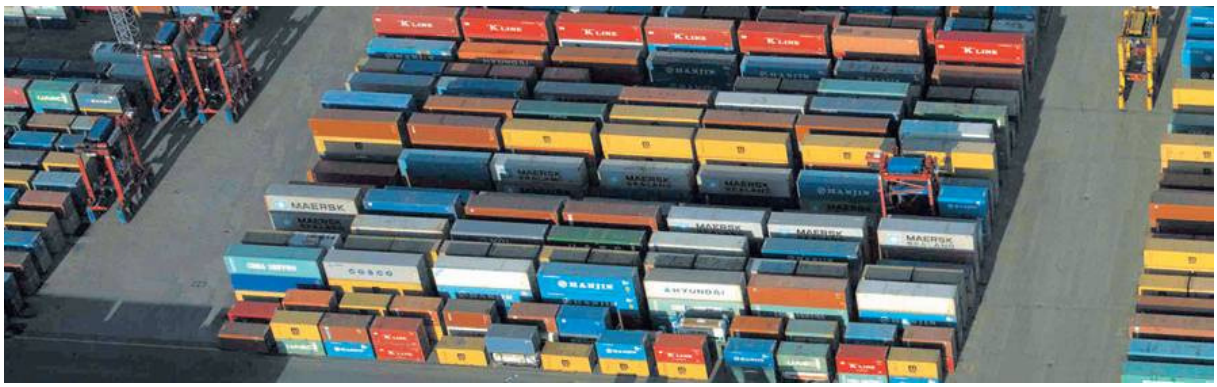


Abbildung 4-2: Aufgereihte Container mit Van Carriern
[Eurogate2006, S. 8]

Ein weiterer Aufgabenbereich, in dem TOPX benötigt wird, ist die Schiffsplanung. In der Schiffsplanung wird entschieden, welcher Container welchem Ladeplatz des Schiffes zugeordnet wird. Auch hier ist eine ganze Reihe an Rahmenbedingungen zu beachten. So ist es nicht der Fall, dass ein voll beladenes Schiff komplett gelöscht und mit anderen Containern beladen wird. Ein Schiff fährt auf seiner Tour verschiedene Häfen an. An jedem Hafen werden Container gelöscht und geladen. Die Container haben entsprechend unterschiedliche Zielhäfen. So muss bei der Schiffplanung darauf geachtet werden, dass die Container, die am

nächsten Zielhafen gelöscht werden sollen, nicht unter Containern stehen, die erst später gelöscht werden. Sonst müssten erst alle Container, die auf dem eigentlichen Löschcontainer stehen, vom Schiff genommen und anschließend wieder verladen werden. Da jede Containerbewegung dem Auftraggeber in Rechnung gestellt wird, kann es im Fall einer Fehlplanung leicht zu Differenzen mit dem Kunden kommen. Ebenfalls sind die verschiedenen Gewichte der Container zu beachten. Das Gewicht sollte möglichst gleichmäßig auf dem Schiff verteilt sein, damit das Schiff nicht in eine Schräglage gerät. Liegt der Schwerpunkt zu weit hinten auf dem Schiff, kann es für die Besatzung der Brücke zu Problemen führe, da die Sichtlinie zum Wasser beeinträchtig wird. Außerdem kann die Stabilität des Schiffes gefährdet werden, wenn eine Querlast entsteht, also das meiste Gewicht z.B. vorne links und hinten rechts liegt. Der Schwerpunkt darf nicht zu hoch oder zu tief liegen, dies hätte Auswirkungen auf das Rollen des Schiffes und könnte in Extremsituationen sogar zum Kentern führen. Ein zu hoher Schwerpunkt kann z.B. dann entstehen, wenn im Laderaum unter Deck viele Leercontainer stehen und über Deck beladene Container verstaut sind.

Abbildung 4-3: Lade- und Löschvorgang am CT Bremerhaven
[Eurogate2007, S. 14]

Ein dritter Aufgabenbereich, in dem TOPX eingesetzt wird, ist die Planung des Geräteeinsatzes. Geräte sind Van Carrier und Brücken. Die Van Carrier können einer Brücke zugeordnet werden, während die Brücke einem Schiff zugeordnet ist. An Bord der Van Carrier befinden sich Handheld-Geräte, die mit TOPX verbunden sind. Über diese Geräte erhält der Van Carrier Fahrer seine Fahraufträge. Ein Fahrauftrag besteht aus einem Quell- und einem Zielstellplatz sowie der dazugehörigen Containernummer. Diese Containernummer ist weltweit eindeutig. Sie besteht aus 4 Buchstaben, 6 Zahlen und einer Prüfziffer. Die ersten 3 Buchstaben stellen das Kürzel des Besitzers dar, während der 4. Buchstabe immer ein „U“ ist. Beginnt eine Containernummer mit „MSCU“, so gehört dieser Container der Reederei „Mediterranean Shipping Company“. Auf den Containern ist diese Nummer an mehreren

Seiten sichtbar angebracht. Mit dieser Containernummer kann der Fahrer visuell prüfen, ob er tatsächlich den richtigen Container bewegt. Die Erstellung dieser Fahraufträge erfolgt im Zuge der Platzoptimierung über TOPX automatisch. Van Carrier, die keiner Containerbrücke zugeordnet sind, werden für die Optimierung der Stellplätze eingesetzt, bei der die Container nur ihre Position auf dem Yard verändern sollen.

Abbildung 4-4: Verladung von Stückgut
[SWOP2007, S.1]

Viele Sonderfälle verkomplizieren diese Abläufe, die hier nur grob skizziert wurden. Es gibt Container, die eine eigene Kühlung besitzen. Diese Kühlcontainer müssen einen Stellplatz erhalten, an dem ein Stromanschluss verfügbar ist. Einige Gefahrgutcontainer müssen gesondert behandelt werden. Es dürfen weder auf dem Yard noch auf einem Schiff gewisse Stoffe nebeneinander gelagert werden. Für Container, die von Insekten befallen wurden, gibt es spezielle Begasungsanlagen. Außerdem können nicht alle Waren mit Containern verschifft werden. Solche größeren Teile, wie zum Beispiel Schiffschrauben, werden als Stückgut verfrachtet. Auf Stückgut kann meistens keine andere Ladung gestapelt werden.

4.2.3 Projektbeschreibung

Im Juni 2008 wurde TOPX beim Containerterminal Hamburg eingeführt und hat das bisherige System um Funktionalitäten erweitert und viele manuelle Ablaufe abgelöst. Es setzt auf den mit COIN erfassten Datenbestand auf. Getestet wurde TOPX auf Integrations- und Systemtestebene sowohl vom Hersteller als auch von einem eigens beauftragten Dienstleister. Trotzdem gab es bei der Einführung Probleme. Ein Resultat daraus ist der Aufbau einer eigenen Testabteilung, die mit den Abläufen an einem Containerterminal vertraut ist und ihre Testdurchführung mit automatisierten Abläufen optimiert. Parallel zu diesem Aufbau der Testabteilung bei Eurogate ITS werden regelmäßig weitere Patches für TOPX implementiert. Diese Patches und neuen Releases müssen getestet werden.

4.2.4 Testanforderung

Um die Funktionsfähigkeit der neuen Releases zu gewährleisten, werden verschiedene Regressionstests ausgeführt. In einer früheren Projektphase wurden hier hauptsächlich Funktionstests ausgeführt, jedoch wurde nicht auf die Korrektheit der Geschäftsabläufe getestet. Dass dabei gravierende Probleme mit den Prozessen übersehen werden, wurde im Test vor der Einführungsphase im Juni 2008 bemerkt. Der Testprozess wurde dahin gehend modifiziert, dass hauptsächlich auf Geschäftsprozesse getestet wurden. Die Einführungsphase ist beendet und es werden verschiedene Änderungsanforderungen umgesetzt und in mehreren Releases umgesetzt. Ziel ist es nun dieses Änderungen zu testen und die betroffenen Bereiche in TOPX mit Regressionstests zu überprüfen.

Das Ziel, das durch die Unterstützung durch Testautomatisierung erreicht werden soll, ist die Durchführung dieser Tests mit weniger Ressourcen. Die Übernahme der Tests vom externen Dienstleister durch Eurogate ITS soll in naher Zukunft erfolgen. Bei ITS sollen aber weniger Leute mit den Tests beauftragt werden, als jetzt von dem Dienstleister eingesetzt werden. Die Übernahme der Tests ist ein aktuell laufender Prozess.

5 Toolauswahl und Bewertung

Ein entscheidender Schritt bei der Einführung einer Testautomatisierung ist die Auswahl der richtigen Werkzeuge. Hierzu stehen im Kapitel 5.1 Hinweise, was bei einer solchen Auswahl zu beachten ist, und Erklärungen dazu, warum diese Auswahl so wichtig für die weitere Arbeit ist. In Kapitel 5.2 wird das HP Quality Center behandelt, welches zu Beginn dieser Arbeit bei Eurogate eingeführt wurde. Die Entscheidung zu dieser Einführung wurde bereits im Vorfeld gefällt und ist nicht Bestand der hier behandelten Toolauswahl, die ab Kapitel 5.3 beginnt.

5.1 Hinweise zur Toolauswahl

Je nach Unternehmen ist die Wahl des Automatisierungstools ein Projekt mit unterschiedlich großen Auswirkungen. Die Entscheidung kann die Arbeit von 2-3 Leuten beeinflussen, die eher experimentell eine Automatisierung erarbeiten – oder den Arbeitstag von mehr als 100 Leuten verändern. Entsprechend weitreichend können auch die Folgen sein, wenn das falsche Tool ausgewählt wurde. Die Produktpalette der verfügbaren Tools geht vom kostenlosen Opensource-Tool bis zur kommerziellen Anwendung, die mehrere tausend Euro pro Lizenz kostet. Es wird deutlich, dass die Toolauswahl ein essentielles Thema ist, wenn eine Automatisierung implementiert werden soll.

5.1.1 Anforderungen an eine Einführung von automatisierten Tests

Eine Anforderung an ein Testtool oder an eine Einführung von automatisierten Tests ist zumeist das Beheben aktueller Testprobleme. Solche Probleme können sein:

- manuelles Testen (Zeitaufwendig, Fehleranfällig)
- keine Möglichkeit für Regressionstests
- Erstellung von Testdaten und Testfällen ist fehleranfällig
- schlechte Testdokumentation
- unzureichende Testüberdeckung
- ineffektives Testen

Nicht alle dieser Probleme können mit einem Tool gleichzeitig gelöst werden, deswegen sollten hier Prioritäten gesetzt werden. Eventuell sollte an dieser Stelle auch bedacht werden, dass der Testprozess überarbeitet werden sollte und dass eine Automatisierung nicht die Lösung der Probleme ist. Ein schlecht organisierter Testprozess wird durch eine Automatisierung nicht besser.
[vgl. Fewster1999 S. 248-256]

Es sollte sichergestellt sein, dass die Situation für die Einführung einer Automatisierung geeignet ist. Folgende Aspekte können ausschlaggebend sein:

- keine großen Umbrüche im Unternehmen
- keine Personalengpässe mit Automatisierung überbrücken. Meistens fällt durch die Implementierung zunächst mehr Arbeit an

- eine Person sollte hauptverantwortlich für die Evaluation und Implementierung sein
- es sollte eine gewisse Unzufriedenheit mit den bisherigen Tests existieren
- Unterstützung durch das Top-Management sollte vorhanden sein

[vgl. Fewster1999 S. 256-257]

5.1.2 Auswahlkriterien

Nach einer erfolgten Marktanalyse sind verschiedene Faktoren für die Bildung einer engeren Auswahl an Tools relevant. Zuerst sind dies technische Kriterien. Das Tool muss die verwendeten Hardwareplattformen, Betriebssysteme und Anwendungen unterstützen. Hierbei sollten, soweit möglich, auch zukünftige Projekte beachtet werden. Bei den Betriebssystemen kann es auch möglich sein, dass das Tool auf einem anderen Betriebssystem läuft als die Anwendung, die getestet werden soll. Für den Einsatz weiterer Betriebssysteme können weitere Kosten entstehen, aber diese Alternative sollte bedacht werden, falls nicht ausreichend viele Testtools für ein Betriebssystem zur Verfügung stehen.

Weitere Kriterien betreffen den Anbieter des Tools. Während einer Evaluationsphase sollte darauf geachtet werden, wie gut der Kontakt zum Anbieter ist und wie schnell und präzise Supportanfragen bearbeitet werden. Der Anbieter möchte mit seinem Produkt Geld verdienen. Wenn das Tool gekauft wurde, also der Anbieter sein Geld bekommen hat, ist es nicht sehr wahrscheinlich, dass sich der Kontakt und Support verbessert. Wichtig ist auch die Liste der Referenzkunden. Wenn man zum Beispiel einer der ersten Käufer des Tools ist, gibt es wenig Erfahrungswerte, die bei der Implementierung helfen. Bei weit verbreiteten Tools kann man unter anderem viele Ratschläge in Fachforen (z.B. www.sqaforums.com) erhalten. Ein weiterer Punkt ist die Bereitschaft des Toolanbieters, sein Produkt weiter zu entwickeln oder gegebenenfalls kundenspezifische Verbesserungen vorzunehmen.

Bei dem Tool sollte darauf geachtet werden, dass die Einarbeitungszeit so kurz wie möglich ist. Es sollte also gut zugänglich sein und eine umfangreiche Hilfe besitzen. Hier können auch Kosten für Schulungen eingespart werden. Außerdem sollte das Tool stabil und zuverlässig funktionieren. Wenn längere Testabläufe automatisiert werden und diese unbeaufsichtigt ablaufen, wäre es sehr hinderlich, wenn das Tool dabei abstürzt. Die Integrationsmöglichkeiten mit anderen Tools kann ein wichtiger Aspekt sein. Zum Beispiel kann die Möglichkeit, Testergebnisse direkt in ein bereits verwendetes Dokumentenmanagementsystem zu schreiben, sehr komfortabel sein.

Welche Faktoren in der Praxis von entscheidender Bedeutung sind, ist im Einzelfall festzulegen. Es bietet sich an, eine Tabelle mit notwendigen und wünschenswerten Features zu erstellen und die verfügbaren Tools anhand dieser Tabelle zu prüfen. Während dieser Evaluation kann sich diese Tabelle durchaus verändern, da Tools Features enthalten können, die man nicht bedacht hat, obwohl sie wünschenswert oder sogar notwendig sind.
[vgl. Fewster1999 S.260-270]

5.2 HP Quality Center

Das HP Quality Center ist ein zentrales Qualitätsmanagement-Tool, mit dem die Aktivitäten im Testprozess abgebildet und alle relevanten Daten verwaltet werden können. Die Informationen werden in den Modulen Releases, Requirements, Business Components, Test

Plan, Test Lab und Defects erfasst. Die Informationen in diesen Modulen sind miteinander verknüpft, so dass der Status des Projektes jederzeit ersichtlich ist. Diese Übersicht wird durch die Möglichkeit, diverse Analysen zu erstellen, noch verbessert. Es können Berichte und Graphen erstellt werden, die über den aktuellen Status oder Verlauf des Projektes Auskunft geben. Sie enthalten unter anderem Informationen über die aktuellen Anforderungen, den Verlauf und Stand der Tests, die Testfallabdeckung, die Anzahl und den Verlauf der Fehlerzustände.

Das Quality Center wird in einem Webbrowser ausgeführt, benötigt allerdings ActiveX-Komponenten, die den Internet Explorer ab Version 6 voraussetzen. Da keine Clientinstallation notwendig ist, kann recht einfach ein unternehmensweiter oder sogar weltweiter Zugriff übers Internet auf die Projekte ermöglicht werden. Die Zugriffsrechte werden über ein Administrationsmodul verwaltet. Es können QC-spezifische Benutzerkonten angelegt oder Daten über LDAP (Lightweight Directory Access Protocol) importiert werden. Die Datenhaltung selbst erfolgt mit Hilfe eines DBMS (Database Management Systems) wie MS SQL-Server oder Oracle.

In [HPQC-Tut2007] wird beispielhaft der Test einer Webseite, die das Buchen von Flügen ermöglicht, dargestellt. Dieses Beispiel wird auch in [HPQTP-Tut2008], einem Tutorial zum Automatisierungstool QuickTest Professional, verwendet. Im Folgenden stammen alle Screenshots aus diesem Beispielprojekt. Die dazugehörige Webseite ist unter http://newtours.demoaut.com/ zu erreichen. HP hat diese Tools von der Firma „Mercury“ übernommen und weiter entwickelt, deshalb taucht der Name „Mercury“ weiterhin an einigen Stellen auf.

5.2.1 Releases

Zu einem Projekt können verschiedene Releases verwaltet werden. Zu jedem dieser Releases können mehrere frei definierbare Cycles angelegt werden. Mit Cycles lassen sich die Releases in kleinere Abläufe gliedern. Diese Art der Unterteilung entspricht dem iterativen Entwicklungsmodell. Zu einem Release kann eine Beschreibung erfasst sowie mehrere Anhänge in Form von Bildern, Dateien, Links etc. angefügt werden. Außerdem können Start- und Enddaten für das Release angelegt werden, so dass eine Aussage über den Fortschritt gemacht werden kann. Die gleichen Daten können für die Cycles erfasst werden. Auf Grundlage dieser Daten und verschiedenen Daten aus den anderen Modulen können Auswertungen über den Status der Releases erstellt werden.

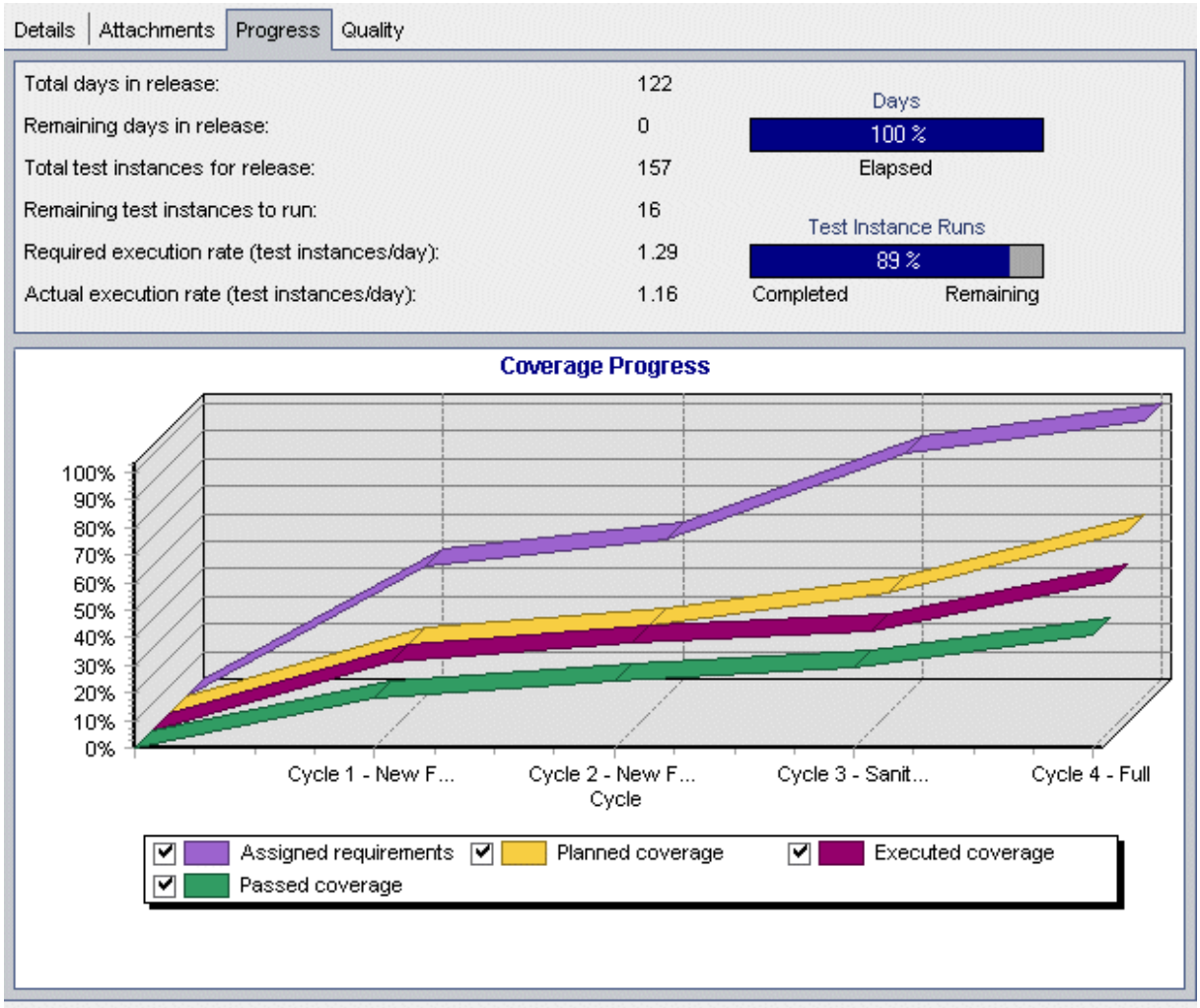


Abbildung 5-1: HP Quality Center Graphen Testfallüberdeckung

Diese Grafik zeigt den Verlauf der Testabdeckung zu dem gewählten Release. Die unterschiedlichen Graphen stellen die zugewiesenen Requirements (Assigned requirements), die geplante Testfallabdeckung (Planned coverage), die Abdeckung mit tatsächlich ausgeführten Tests (Executed coverage) und die Abdeckung mit erfolgreichen Tests (Passed coverage) dar. Die Zeitachse wird durch die verschiedenen Cycles gegliedert.

Zur Unterstützung des Defect Trackings kann über den Reiter „Quality“ eine Auswertung aufgerufen werden, die den zeitlichen Verlauf der Defects darstellt. Die Defects sind nach ihrem Schweregrad gruppiert. Es gibt eine Auswertung über die Anzahl noch offener Defects und über die Anzahl an Defects, die im Verlauf des Releases oder Cycles aufgetreten sind.

5.2.2 Requirements

Das Requirements Modul dient zur Verwaltung der Anforderungen an das Softwareprojekt. Die Anforderungen können in funktionale und verschiedene nicht funktionale Anforderungen gegliedert werden. Sie werden in einer Baumstruktur organisiert und dargestellt.

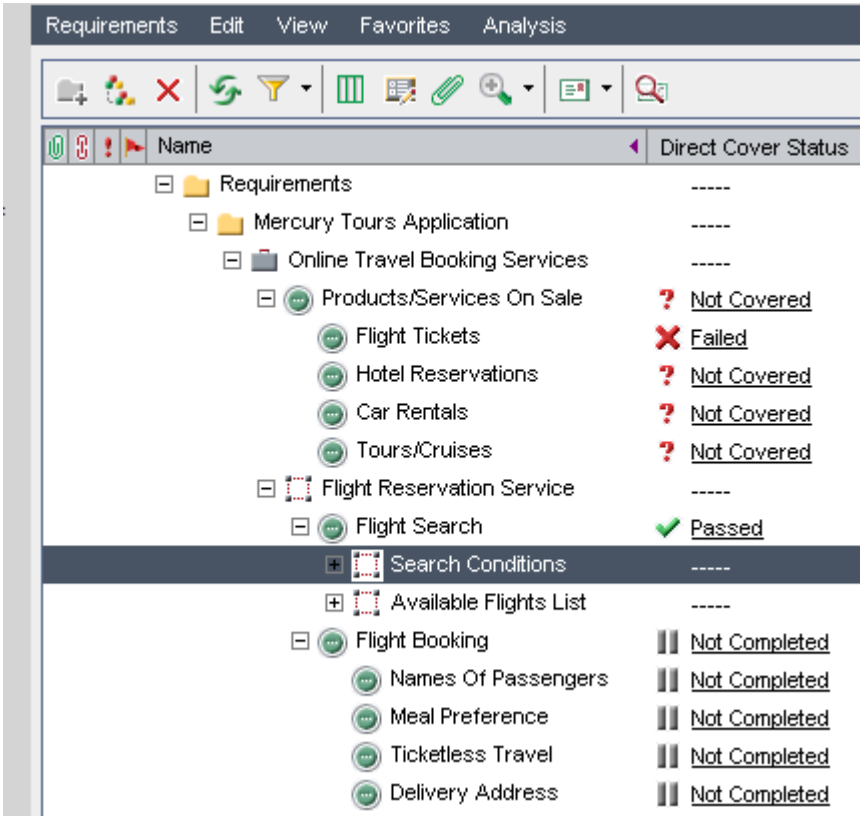


Abbildung 5-2: HP Quality Center Requirements

Neben der Zuordnung zu den Releases und Cycles, zu denen die Anforderungen gehören, erhält man auch eine Übersicht über die Abdeckung mit Testfällen. Es gibt die Kategorien „Not Covered“ (Es wurde kein Testfall zugewiesen), „Not Completed“ (Nicht alle zugewiesenen Testfälle wurden ausgeführt), „Failed“ (mindestens ein zugewiesener Testfall hat einen Fehler aufgedeckt) und „Passed“ (alle zugewiesenen Testfälle wurden erfolgreich durchgeführt). Der jeweilige Status wird durch ein Symbol gekennzeichnet.

Außerdem bietet das Quality Center eine Unterstützung für die Risikoabschätzung. Die Risiken werden in Geschäftsrisiken (Business Criticality) und Fehlerwahrscheinlichkeit (Failure Propability) unterteilt. In beiden Kategorien kann man verschiedene Bewertungen abgeben. Bei den Geschäftsrisiken können Abschätzungen dazu abgegeben werden, wie oft die entsprechende Funktion verwendet wird, wie viele Benutzer von einem eventuellen Fehler betroffen sind und wie hoch der Schaden im Fehlerfall wäre. Aus diesen Angaben berechnet das Quality Center einen Vorschlag zur Einstufung in die Kategorien A, B oder C.

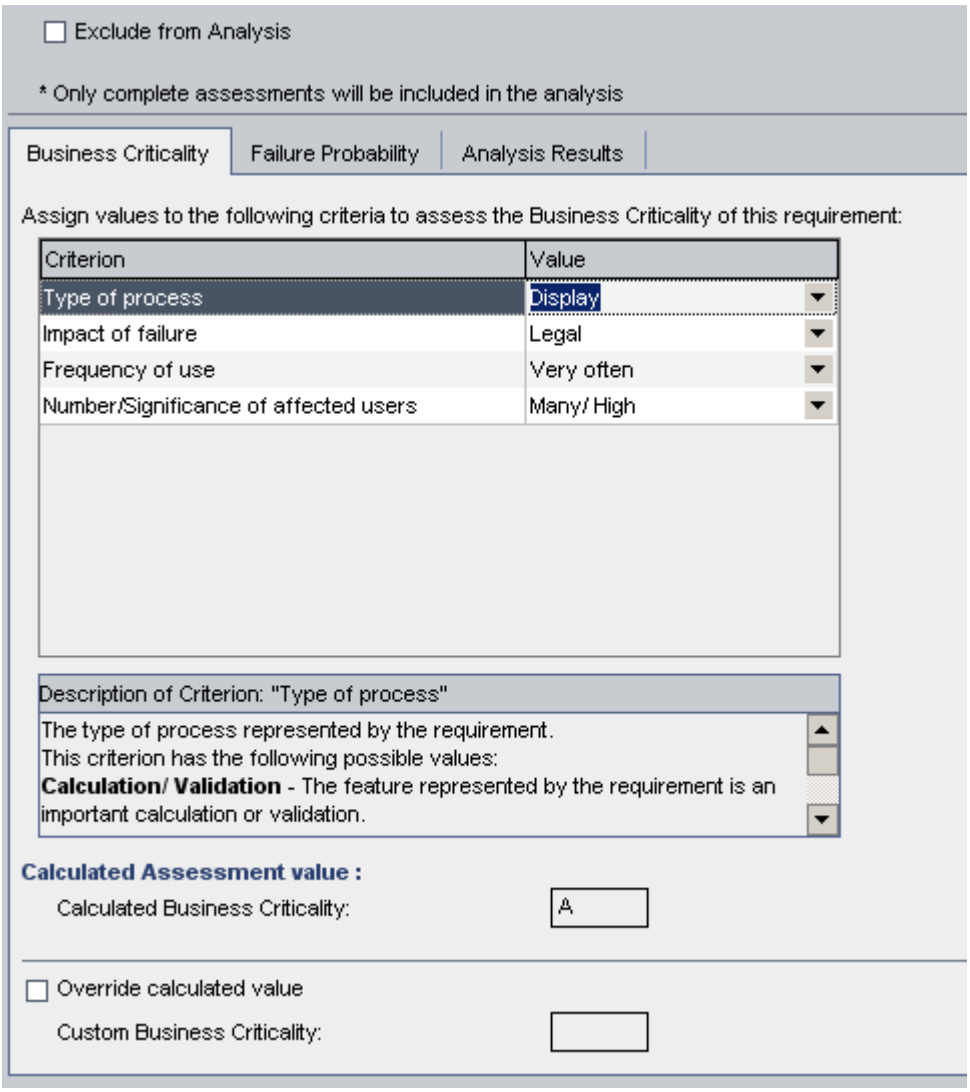

Exclude from Analysis
* Only complete assessments will be included in the analysis
Business Criticality
Failure Probability
Analysis Results
Assign values to the following criteria to assess the Business Criticality of this requirement:
Criterion
Value
Type of process
Display
Impact of failure
Legal
Frequency of use
Very often
Number/Significance of affected users
Many/ High
Description of Criterion: "Type of process"
The type of process represented by the requirement.
This criterion has the following possible values:
Calculation/ Validation - The feature represented by the requirement is an important calculation or validation.
Calculated Assessment value :
Calculated Business Criticality:
A
Override calculated value
Custom Business Criticality:

Abbildung 5-3: HP Quality Center Risikoanalyse

Die Einstufung der Fehlerwahrscheinlichkeit in die Kategorien 1, 2 und 3 erfolgt anhand der Angaben zur Änderungsart (neue Anforderung, Änderungen, keine Änderung), Reifegrad der Software, Anzahl der bekannten Fehler und der Anzahl der betroffenen Programmteile.

5.2.3 Business Components

Das Business Components Modul ist eine optionale Erweiterung von HP für das Quality Center, die nicht in jeder Ausbaustufe enthalten ist. Es wird eine extra Lizenz benötigt. Das Modul dient zur Modellierung von Geschäftsabläufen durch Experten der entsprechenden Fachabteilungen. Hier können Abläufe beschrieben werden, auf deren Grundlage Tests erstellt werden. Dies kann bereits vor der Fertigstellung der Software geschehen. In einem späteren Schritt kann der Testautomatisierer die beschriebenen Abläufe zum Beispiel in QuickTest Professional automatisieren. Hierfür ist eine Schnittstelle sowohl in QuickTest Professional als auch im Quality Center vorgesehen. Das Automatisierungsskript kann im Quality Center eingesehen und editiert werden.

Die im Business Components Modul erzeugten Tests können nach ihrer Vervollständigung als „Business Process Test“ im Test Plan Modul eingebunden und, analog zu den anderen Testarten im Test Lab, ausgeführt werden.

5.2.4 Test Plan

Das „Test Plan“ Modul wird zum Erstellen der Testfälle benutzt. Die Tests können in einer Ordnerstruktur organisiert werden, die mit einer Baumansicht auf der linken Seite dargestellt wird. Es können verschiedene Arten an Tests angelegt werden. Dies sind zum Beispiel manuelle Tests, Business Process Tests oder QuickTest Professional Tests. Zu jedem Test werden zusätzliche Daten wie Priorität und Status erfasst.

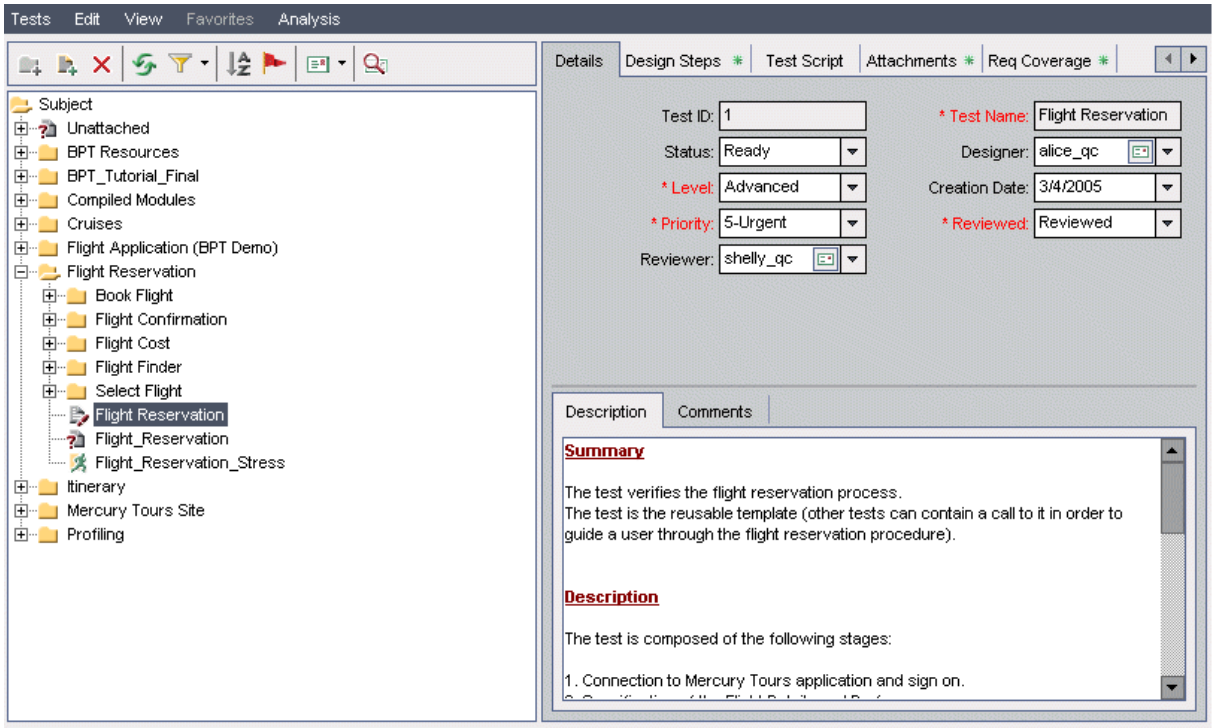


Abbildung 5-4: HP Quality Center Test Plan

Bei manuellen Tests werden unter dem Reiter „Design Steps“ die durchzuführenden Testschritte erstellt. Dort wird hinterlegt, was genau gemacht werden soll und welches Ergebnis erwartet wird. Bei automatischen Tests werden Testskripte hinterlegt. Des Weiteren werden die Tests mit den Requirements, die sie abdecken, und eventuell mit Defects verlinkt. Diese Verlinkungen sind wichtig, um mit Hilfe der diversen Auswertungen den Überblick über das Projekt zu behalten.

5.2.5 Test Lab

Im „Test Lab“ werden die Tests, die im „Test Plan“ erstellt wurden, zu „Test Sets“ zusammengefügt. Diese so erstellten Sets können dann im „Test Lab“ auch ausgeführt werden. Die Test Sets können wie gewohnt in einer Baumstruktur geordnet werden. Um die gewünschten Tests auszuwählen, kann man entweder den „Test Plan“-Baum oder den „Requirements“-Baum einblenden – je nach dem, ob man Tests zu bestimmten Requirements

sucht oder in der Struktur des „Test Plan“ schneller fündig wird. Die Test Sets können wieder mit Defects verknüpft werden.

Mit dem „Execution Flow“ können Bedingungen erstellt werden, die bestimmen, wann und ob welcher Test ausgeführt werden soll. Diese Bedingungen können Datum und Uhrzeit sein oder der Status eines vorherigen Tests. So kann man zum Beispiel festlegen, dass ein Test nur ausgeführt wird, wenn einer oder mehrere vorherige Tests erfolgreich abgeschlossen wurden. Diese Abhängigkeiten lassen sich mit Drag and Drop erstellen. Das Ergebnis ist ein Diagramm, wie in Abbildung 4-6 gezeigt.
.

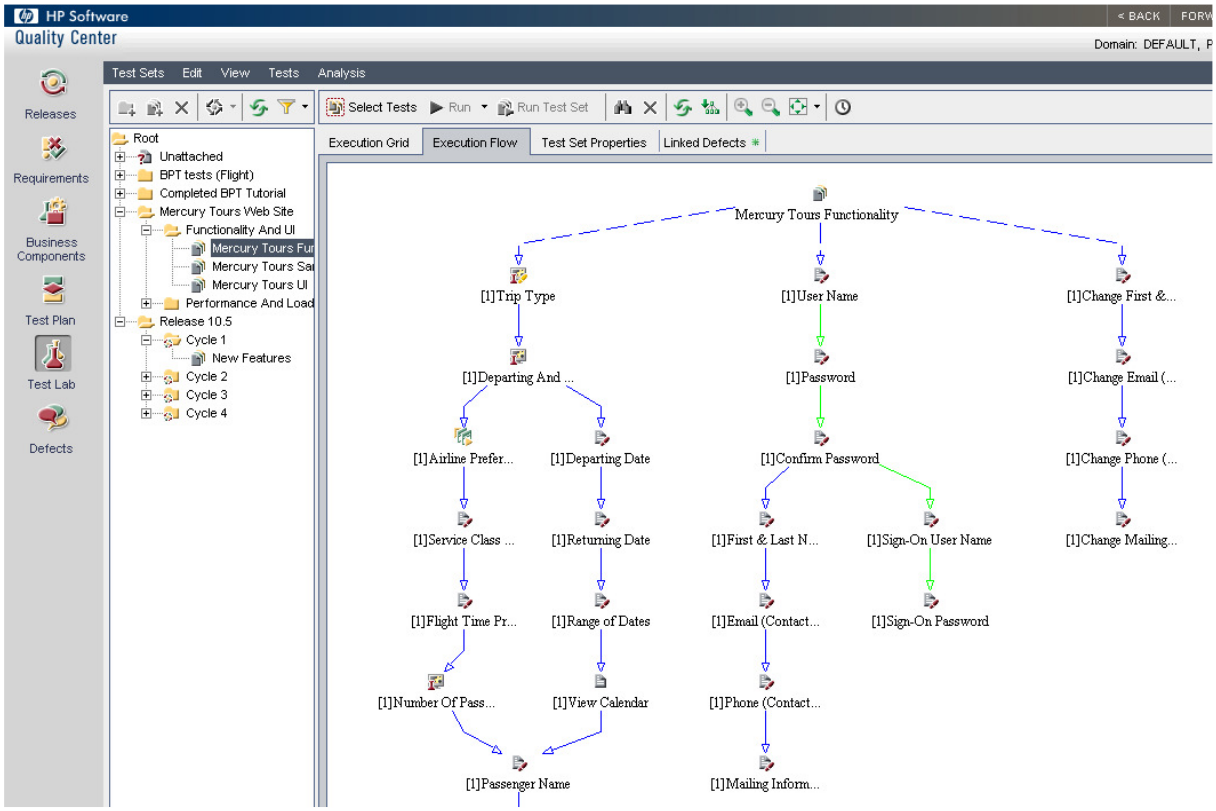


Abbildung 5-5: HP Quality Center Test Lab Execution Flow

Der gestrichelte Pfeil bedeutet, dass der Test ausgeführt wird, sobald der vorherige Test gelaufen ist. Ein blauer, durchgezogener Pfeil bedeutet, dass der Test ausgeführt wird, wenn der vorherige Test mit „finished“ beendet wurde. Das Ergebnis „finished“ bedeutet, dass der Test nicht aufgrund eines Fehler abgebrochen wurde. Der grüne Pfeil bedeutet, dass der Vorgänger nicht nur beendet, sondern auch erfolgreich abgeschlossen (passed) werden muss, damit sein Nachfolger ausgeführt wird. Eine kleine Uhr am Pfeil bedeutet, dass der nachfolgende Test nur zu einer bestimmten Uhrzeit oder an einem bestimmten Datum ausgeführt werden soll. Dies kann sinnvoll sein, wenn länger dauernde Tests über Nacht automatisch ablaufen sollen.

Bei der Ausführung von manuellen Tests werden dem Tester die Testschritte nacheinander in einem Fenster angezeigt. Zu den Testschritten sind die erwarteten Ergebnisse dokumentiert, es liegt nun an dem Tester zu entscheiden, ob das tatsächliche Verhalten der Software mit dem erwarteten Verhalten übereinstimmt. Bei einer Ausführung von automatisierten Tests kann zum Beispiel QuickTest Professional vom Quality Center aufgerufen werden. Es wird

auf dem spezifizierten Testrechner gestartet, und der Test wird ausgeführt. Nach Abschluss des Tests werden die Ergebnisse im Quality Center angezeigt.

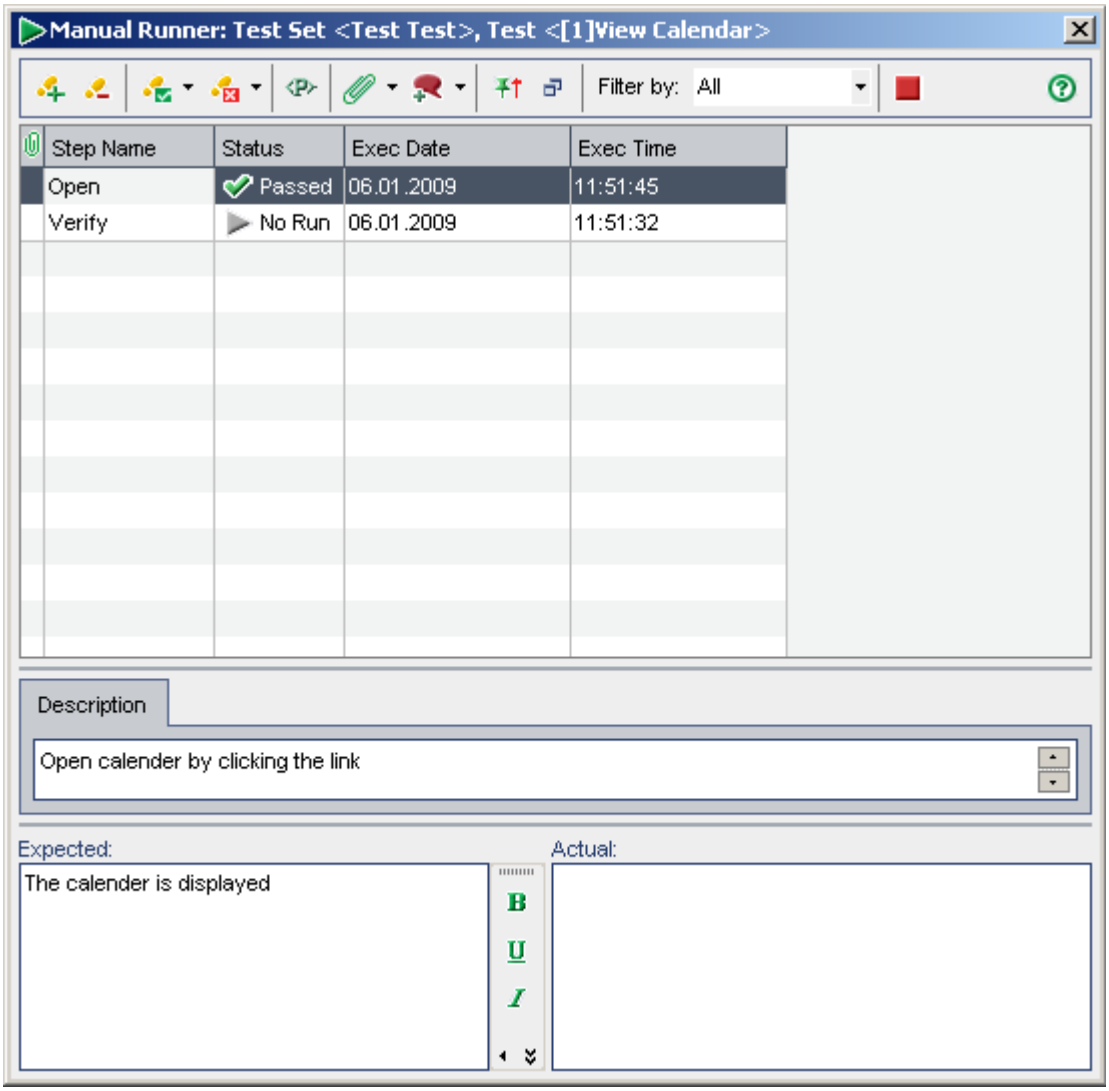


Abbildung 5-6: HP Quality Center Test Lab manuelle Testausführung

Es besteht auch die Möglichkeit, Test Sets aus manuellen und automatisierten Tests zu kombinieren. So können eventuell vorbereitende Schritte für einen Test automatisiert werden. Hierbei kann es sich z.B. um das Anlegen von bestimmten Datensätzen handeln. Der eigentliche Test benötigt diesen bestimmten Datensatz, kann aber aus irgendwelchen Gründen nicht automatisiert werden. So wird dem Tester aber zumindest das Anlegen der Testdaten abgenommen. Ein anderes Beispiel für eine Kombination aus manuellem und automatisiertem Test wäre das Überprüfen eines Ausdrucks auf Papier. Hier muss eine manuelle Prüfung erfolgen.

5.2.6 Defects

Im Defects Modul werden alle aufgedeckten Fehler der Software dokumentiert. Die Fehler können mit der Testausführung verlinkt werden, in der er aufgetreten ist. Da dieser Test

wiederum mit den entsprechenden Requirements verlinkt ist, kann eine Auswertung darüber gemacht werden, welches Requirement aufgrund eines bestimmten Fehlers nicht erfüllt wird. Fehler können außerdem mit anderen Fehlern, mit Anforderungen oder Testfällen verlinkt werden.

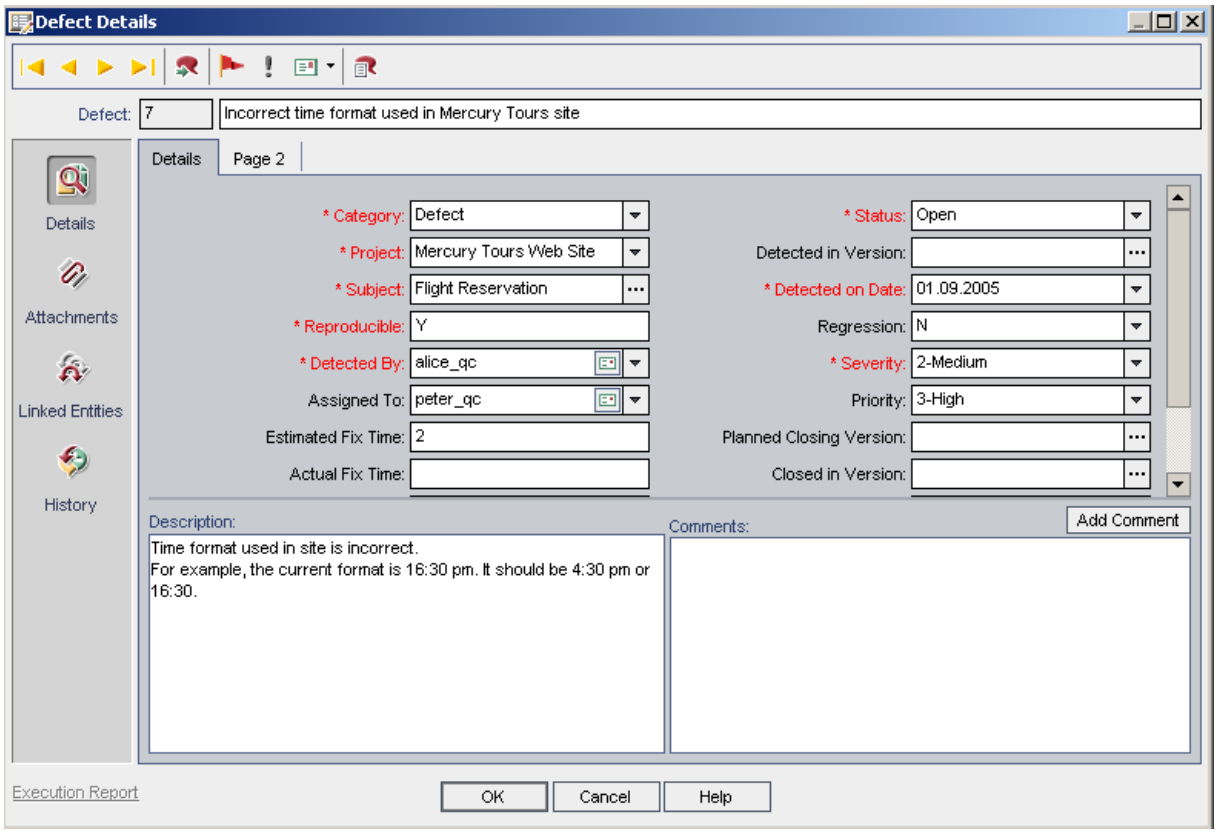


Abbildung 5-7: HP Quality Center Defects

Um die Nachvollziehbarkeit eines Fehler zu erhalten, wird vom Quality Center eine Historie gepflegt, in der alle Änderungen, die während des Projektverlauf vorgenommen wurden, dargstellt werden.

5.2.7 Zusammenfassung

Das Quality Center bildet eine Grundlage für eine umfassende Dokumentation kompletter Softwareprojekte. Viele Eingabefelder können über ein Administrationstool für jedes Projekt individuell angepasst werden. Hier lassen sich mögliche Eingaben und Pflichteingaben festlegen. Die Möglichkeit, automatisierte Tests mit verschiedenen Tools vom Quality Center auszuführen, sowie die Kombinationsmöglichkeit mit manuellen Testabläufen bieten eine hohe Flexibilität. Damit ist es möglich, jede Art von Applikation mit dem Quality Center zu verwalten. Die Verzahnung der verschiedenen Module bildet eine Grundlage für sehr viele Auswertungen, die alle benötigten Metriken bereitstellen, um einen Überblick über den Projektfortschritt und den Testprozess zu verschaffen.

Bei Eurogate werden in der Phase der Einführung des Quality Centers alle Daten zu kommenden Releases und den damit verbundenen Requirements von TOPX erfasst. Die Erfassung der Defects erfolgt weiterhin in einer selbstprogrammierten Webapplikation. Auf

diese Daten hat der Hersteller von TOPX Zugriff und kann diese direkt in seinen Entwicklungsprozess einfließen lassen. Für die COIN-Migration ist geplant, die Defects ebenfalls im Quality Center zu verwalten und dem Lieferanten des Neusystems auf diesen Teil des Projektes den Zugriff zu ermöglichen, um eine effektive Kommunikation und somit eine zeitnahe Fehlerkorrektur zu ermöglichen.

5.3 Auswahl der Automatisierungstools

Für die Projekte TOPX und COIN sollen Tools zur Automatisierung ausgewählt werden. Zuerst ist es notwendig, eine Kriterienmatrix aufzustellen, anhand derer die verfügbaren Tools bewertet werden können. Die Tools sollen alle von einem Windowsrechner aus bedient werden und müssen für TOPX mit einer auf QT basierten Oberfläche arbeiten können. TOPX selbst wird auf Sun Solaris ausgeführt. COIN wird auf einem Unisys Großrechner über eine Hostemulation ausgeführt. Hier wird es unwahrscheinlich sein, ein Tool zu finden, dass ebenfalls auf dem Unisys-Rechner läuft. Deshalb wird es notwendig sein, dass das Automatisierungstool auf Windows ausgeführt wird und über den Hostemulator mit COIN kommuniziert. Die erstellten Tests sollten über das HP Quality Center verwaltet werden können, um eine einheitliche Dokumentation zu ermöglichen. Sehr vorteilhaft wäre es, wenn man für beide Projekte das gleiche Tool verwenden könnte.

Folgende Kriterien wurden identifiziert und in einer Tabelle dargestellt. Die Priorität wird in mit 1 oder 2 festgelegt. 1 = notwendig, 2 = wünschenswert.

ID	Kriterium	Beschreibung	Priorität
1	Integration in das HP Quality Center	Die Testfälle, die mit dem Tool generiert werden, sollen mit dem Quality Center verwalten werden können.	1
2	Objekterkennung von QT-Objekten	Das Tool sollte in der Lage sein, Objekte des QT-Toolkits zu erkennen.	1
3	Objekterkennung von Windows-Objekten	Nicht zwangsweise notwendig, kann aber für spätere Projekte eventuell vom Vorteil sein.	1
4	Skriptsprache	Es sollten geläufige oder leicht erlernbare Skriptsprachen unterstützt werden, die einen entsprechenden Funktionsumfang besitzen.	2
5	Datenbank Zugriff	Kann das Tool auf externe Datenbanken zugreifen? Für datengetriebene Tests oder Verifikation von Dateninhalten.	2
6	Lasttest	Kann mit dem Tool ein Lasttest durchgeführt werden? Einige Tools benötigen dafür ein kostenpflichtiges Add-On.	2

7	Betriebssystem	Auf welchem Betriebssystem wird das Tool ausgeführt? Kann es einfach auf einem Windowsrechner betrieben werden, oder werden weitere Systeme oder Terminalemulationen benötigt?	2
8	Kosten / Lizenz	Wie teuer ist das Tool pro Lizenz?	2

Nach der Markanalyse per Internet (Hersteller-Homepage, Fachforen) und aufgrund von Erfahrungsberichten von Kollegen wurden folgende Tools anhand der oben genannten Kriterien bewertet. Ziel war es, Kandidaten für eine engere Auswahl zu ermitteln:

	AutomatedQA Testcomplete	**Froglogic Squish**
Integration HP QC	Nein	ja, mit Add-In
Objekterkennung QT	Nein	ja
Objekterkennung Windows	Ja	nein
Skriptsprache	VBScript, JScript, Delphi Script, C++ Script	JScript, Tcl, Python
Datenbank Zugriff	Ja	ja
Lasttest	nein	ja
Betriebssystem	Windows	Windows, Linux, MacOS, UNIX
Kosten / Lizenz	ca. 3.000 EUR	ca. 2.400 EUR

	HP QuickTest Professional	**ICS Replay Xcessory**	**Redstone Software Eggplant**
Integration HP QC	ja	nein	bedingt möglich
Objekterkennung QT	nein	nein, nur Motif	ja, über Bilderkennung
Objekterkennung Windows	ja	nein	ja, über Bilderkennung
Skriptsprache	VB Script	Tcl	Sense Talk
Datenbank Zugriff	ja		nein (geplant)
Lasttest	nein		nein
Betriebssystem	Windows	UNIX	MacOS
Kosten / Lizenz	ca. 3.000 EUR (2 Lizenzen bereits vorhanden)	k.a.	ca. 3.750 EUR

Für den GUI-Test von TOPX wird eine Erkennung von QT-Elementen zwingend vorausgesetzt. Aufgrund dessen bleiben nur die Tools Squish und Eggplant übrig. Beide Tools haben Vor- und Nachteile, die mit Evaluationsversionen genauer betrachtet wurden.

QuickTest Professional ist leider nicht in der Lage, mit QT-Elementen zu arbeiten. Da aber bereits 2 Lizenzen bei Eurogate vorhanden sind, wird der Einsatz für den Test von COIN bewertet. Es folgt eine Beschreibung der drei Tools sowie eine Bewertung. Abschließend wird das Ergebnis der Toolauswahl dargestellt.

5.4 HP Quicktest Professional

HP Quicktest Professional (QTP) ist ein Tool zur Automatisierung von GUI-Tests unter Windows. Die internen Repräsentationen von Windows-Steuerelementen können erkannt und verwendet werden. Durch Plug-Ins ist es unter anderem auf SAP- und Javaanwendungen erweiterbar. Das Aufzeichnen der Testabläufe wird mittels Capture&Replay realisiert. Da QTP von HP entwickelt wird, wird auch eine gute Anbindung an das Quality Center angeboten. Testfälle können hier abgelegt, verwaltet und ausgeführt werden.

5.4.1 Skripterstellung

Die Application under Test (AUT) wird auf dem Rechner ausgeführt, auf dem auch QuickTest Professional gestartet wurde. Dann wechselt man in einen Capture-Modus und führt den geplanten Testfall in der AUT aus. Alle Benutzeraktionen werden von QTP erkannt und im Skript gespeichert. Mit diesem Skript kann der Testfall automatisch wiederholt werden. Als Skriptsprache steht VBScript zur Verfügung. Allerdings kann QTP auch in andere Entwicklungsumgebungen wie Microsoft Visual Studio eingebunden werden. Somit stehen dann alle Sprachen dieser Entwicklungsumgebung bereit.

Die einzelnen Tests sind in „Actions“ untergliedert, die einzelne Testschritte darstellen. Es ist möglich, diese „Actions“ auch in andere Tests einzubinden. So müssen häufig verwendete Schritte nicht jedes Mal neu aufgezeichnet werden. Falls sich an diesen Schritten etwas ändert, weil vielleicht etwas an der AUT angepasst wurde, müssen diese Änderungen an der „Action“ nur einmal durchgeführt werden; an die referenzierten Actions werden die Änderungen weitergegeben. Neben den „Actions“ können auch Funktionsbibliotheken projektübergreifend verwenden werden. Dies steigert die Wartbarkeit der Testfälle.

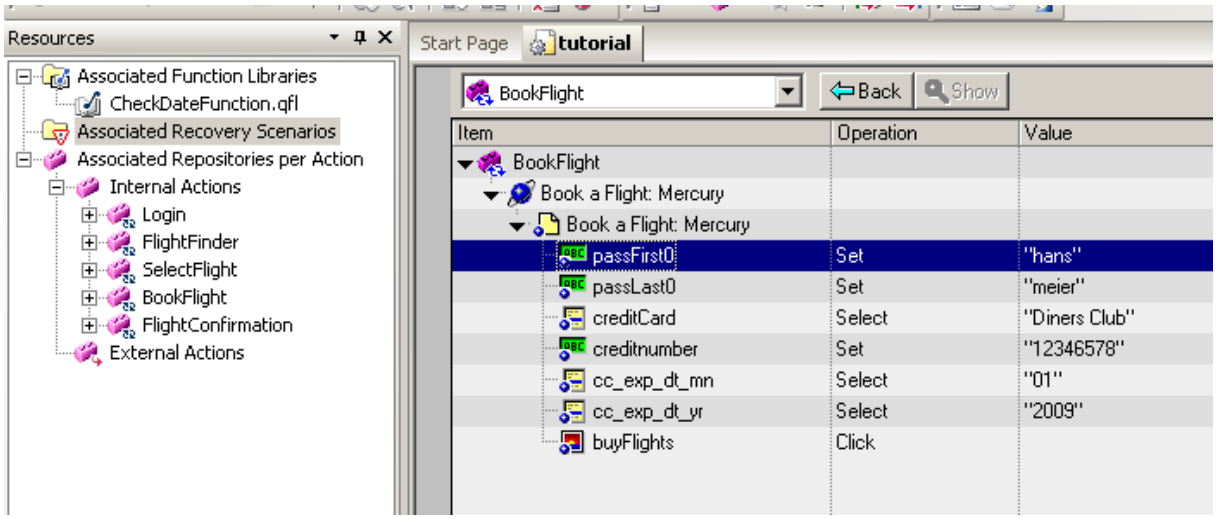


Abbildung 5-8: HP QuickTest Professional Action im Keyword View

Auf dem Screenshot sind im linken Bereich eine verlinkte Funktionsbibliothek und die „Action“ zu sehen, die der Reihe nach abgearbeitet werden. Rechts erkennt man die Details zu der „Action“ „BookFlight“. Bei der AUT handelt es sich in diesem Fall um eine Webapplikation, die von HP für das Tutorial zu QTP [HPQTP-Tut2008] bereit gestellt wird.

In dem Tutorial sind die wichtigsten Grundfunktionalitäten von QTP erläutert. Die Page ist unter http://newtours.demoaut.com/ zu erreichen. Diese Webanwendung ist nicht mit den in dieser Arbeit vorgestellten Projekten vergleichbar, aber um die Arbeitsweise von QuickTest Professional darzustellen, ist diese Anwendung sehr gut geeignet.

Die Sicht auf das Automatisierungsskript ist umschaltbar zwischen dem „Keyword View“ und dem „Expert View“. Auf der Abbildung ist der „Keyword View“ zu sehen. In diesem können die einzelnen Testschritte per Mausklick erstellt werden und werden entsprechend grafisch dargestellt. Dieser Modus richtet sich an die Anwender mit wenig Programmiererfahrung. Im „Expert View“ wird direkt mit VBScript gearbeitet. In beiden Ansichten können die gleichen Ergebnisse erzielt werden. Welche bevorzugt wird, hängt von der Anwendung ab, die getestet werden soll, außerdem vom Erfahrungshorizont sowie den Präferenzen des Anwenders. Dies ist das Skript zu der in 5.8 dargestellten „Action“:

```
Browser("Book a Flight: Mercury").Page("Book a Flight:
Mercury").WebEdit("passFirst0").Set "hans"

Browser("Book a Flight: Mercury").Page("Book a Flight:
Mercury").WebEdit("passLast0").Set "meier"

Browser("Book a Flight: Mercury").Page("Book a Flight:
Mercury").WebList("creditCard").Select "Diners Club"

Browser("Book a Flight: Mercury").Page("Book a Flight:
Mercury").WebEdit("creditnumber").Set "12346578"

Browser("Book a Flight: Mercury").Page("Book a Flight:
Mercury").WebList("cc_exp_dt_mn").Select "01"

Browser("Book a Flight: Mercury").Page("Book a Flight:
Mercury").WebList("cc_exp_dt_yr").Select "2009"

Browser("Book a Flight: Mercury").Page("Book a Flight:
Mercury").Image("buyFlights").Click
```

5.4.2 Verifikation der Ergebnisse

Bei der Testautomatisierung ist es nicht nur wichtig, dass die Tests automatisch ausgeführt werden, es ist auch unerlässlich, dass die erzeugten Ausgaben automatisch verifiziert werden können.

In das Automatisierungsskript können verschiedene Prüfungen eingebaut werden. Es können „Checkpoints“ verwendet werden, die den Zustand von Steuerelementen, Werte in Variablen oder Daten in Datenbanken auf ihre Gültigkeit prüfen. Ungültige Werte wären Abweichungen von den erwarteten Werten.

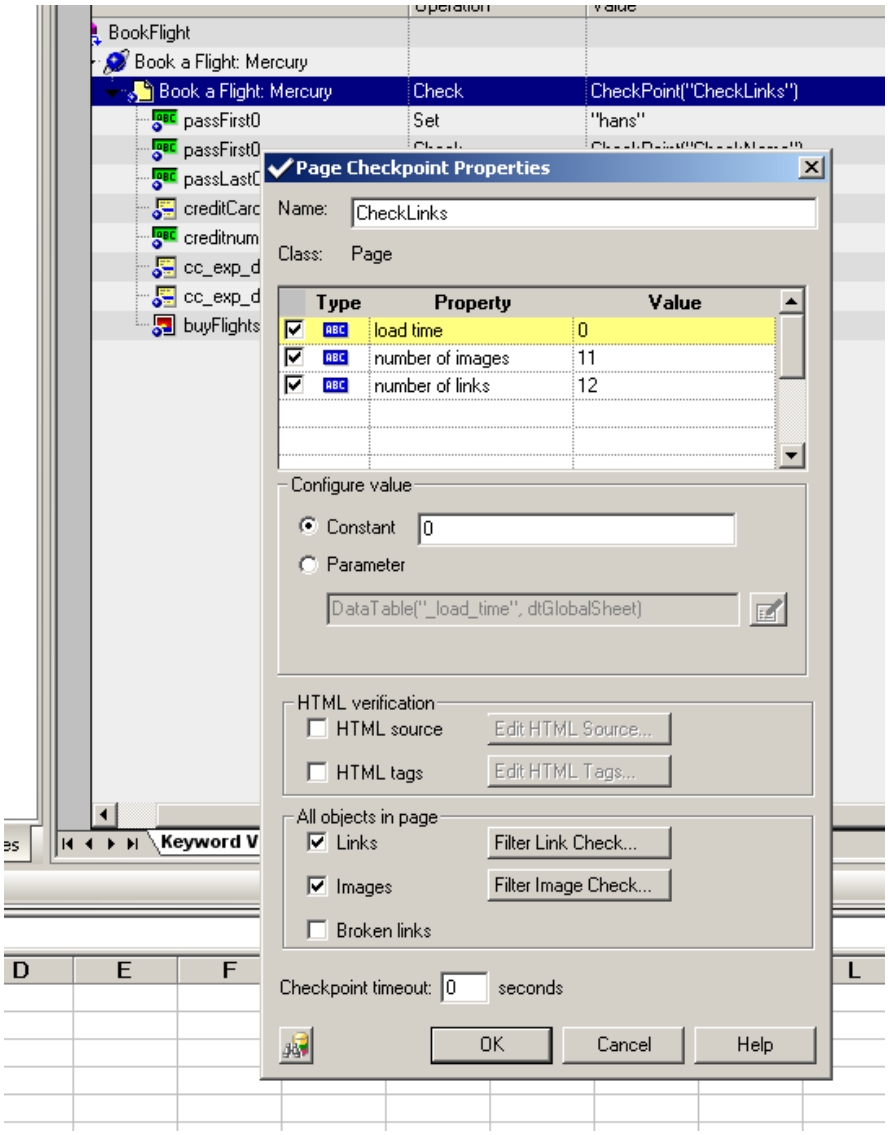


Abbildung 5-9: HP QuickTest Professional Checkpoint

In diesem Beispiel aus dem Tutorial wird überprüft, ob auf der geladenen Webseite 11 Bilder und 12 Links angezeigt werden. Um zu verhindern, dass ein langsamer Aufbau der Seite dazu führt, dass nicht sofort alle Bilder angezeigt werden, kann in das Feld „load time“ eine Zeit eingetragen werden, nach der die Seite fertig aufgebaut sein muss.

Es besteht auch die Möglichkeit, per Quellcode eigene Abfragen zu generieren und gegebenenfalls einen Fehler zu melden. Alle Fehler, die während eines Testlaufs aufgetreten sind, werden nach der Testdurchführung in einem Berichtsfenster angezeigt. Im Automatisierungsskript können mit dem Befehl *reporter.ReportEvent* Fehler, Warnungen oder Erfolgsmeldungen erzeugt werden. Diese erhalten noch einen Namen und eine genauere Beschreibung und werden somit nach der Testdurchführung angezeigt. Bei Fehlern, die mit Hilfe der Checkpoints aufgedeckt wurden, werden alle überprüften Daten angezeigt, und es wird ein Screenshot der Anwendung zu dem entsprechenden Zeitpunkt dargestellt.

5.4.3 Datengetriebener Test

In der „Data Table“ können Daten zu jeder „Action“ und globale Daten für das Testprojekt angelegt werden. Die „Data Table“ ist nichts anderes als eine Excel Tabelle, die in QTP eingebunden ist. Diese Tabelle liegt im Projektverzeichnis und kann auch mit Excel bearbeitet werden. Die Spaltennamen sind Namen von Parametern, die in den Testschritten verwendet werden können. Somit stellt jede Zeile in der Tabelle einen Datensatz dar. Man kann zum Beispiel die Eingabe in ein Textfeld mit Hilfe dieser Parameter erledigen. Trägt man nun in die Tabelle mehrere mögliche Eingaben in die Zeilen ein, kann man QTP so konfigurieren, dass für jeden Datensatz ein neuer Durchlauf des Testschrittes erfolgen soll.

5.4.4 Zusammenfassung

Mit Quicktest Professional lassen sich recht schnell Automatisierungsskripte erzeugen. Die Erkennung von Objekten unter Windows erfolgt zuverlässig, und das Capture and Replay führt schnell zu brauchbaren Ergebnissen. Voraussetzung ist natürlich, dass die zu testende Anwendung eine Windows-, Java- oder Webanwendung ist. Es gibt auch für bestimmte Terminalclients eine Unterstützung. Diese konnte jedoch nicht zur Ausführung gebracht werden. Es wurde versucht, die unixbasierte QT-Anwendung TOPX, die mittels „Hummingbird Exceed“ auf einem Windowsrechner ausgeführt wird, mit QTP zu automatisieren. Das Terminalprogramm konnte von QTP nicht erkannt werden, und der Support von HP gab an, dass es keinerlei Unterstützung von Unixanwendungen gibt.

Die Integration in das Quality Center erfolgt problemlos. QTP kann beim Start bereits eine Verbindung mit dem QC aufbauen und somit seine Daten gleich in den entsprechenden QC Projekten ablegen. Die Testfälle erscheinen dann im „Test Plan“ und können anschließend im „Test Lab“ zu komplexeren Testabläufen modelliert werden. Auch die Ausführung der Tests ist direkt über QC möglich. Die Tests können lokal oder remote auf einem Testrechner ausgeführt werden. Auf dem Zielrechner muss jedoch QTP installiert sein. Die Ergebnisse der Testausführung werden im QC gespeichert. Bei auftretenden Fehlern können hier auch die Verlinkungen zu Defects erzeugt werden.

Die implementierte Unterstützung der datengetriebenen Tests mit Hilfe der Excel-Tabelle kann sehr viel Zeit ersparen. Sofern man die Testschritte geschickt parametrisiert hat, können hier sehr viele Tests ausgeführt werden, indem lediglich neue Zeilen zu der Excel-Tabelle hinzugefügt werden. Eine Beispielanwendung wäre die Automatisierung einer Bestellsoftware. Die Bestellung eines Artikels ist immer gleich, nur dass eine andere Artikelnummer eingegeben werden muss. Nun sollte ein Bestellvorgang automatisiert werden, bei der die Artikelnummer ein Parameter ist, der zur Laufzeit aus der Excel-Tabelle ermittelt wird. Durch die Ausführung mehrerer Iterationen werden nun so viele Artikel bestellt, wie Artikelnummern in der Excel-Tabelle hinterlegt sind.

In Hinblick auf den möglichen Einsatz von QuickTest Professional für Tests mit COIN wurde die Zusammenarbeit mit der Hostemulation Exceed getestet. Der Maskeninhalt der textbasierten COIN-Masken lässt sich problemlos als String auslesen, und Tastatureingaben können ohne Aufwand an das Exceed-Fenster gesendet werden. Für die Positionierung des Cursors in bestimmten Eingabefeldern können Mausklicks auf Fensterkoordinaten verwendet werden. Da nicht zu erwarten ist, dass das Maskenlayout verändert wird, stellt diese Technik keine Einschränkung dar.

5.5 Redstone Software Eggplant

Bei der Applikation „Eggplant“ handelt es sich um ein Capture and Replay-Tool, welches mit einer Bilderkennung arbeitet. Während Tools wie Quicktest Professional auf die interne Repräsentation der Steuerelemente zugreifen, findet Eggplant die entscheidenden Komponenten anhand gespeicherter Bildausschnitte. Dies ist vor allem dann sinnvoll, wenn man nicht auf die interne Repräsentation zugreifen kann, wie es zum Beispiel bei einer Terminalsession der Fall ist, da dem Clientrechner meistens lediglich ein Bild der Oberfläche zur Verfügung steht. Nähere Informationen über die Art der Objekte oder gar die interne Repräsentation können so nicht erfasst werden. Die Idee von Eggplant ist, dass jede Applikation getestet werden kann, und dies plattform- und technologieunabhängig. Die einzige Voraussetzung ist eine grafische Benutzeroberfläche, die auf einem Rechner im Netzwerk ausgeführt wird, auf den per VNC-Verbindung zugegriffen werden kann.

5.5.1 Skripterstellung

Eggplant wird unter MacOS ausgeführt und stellt eine Verbindung über einen eigenen VNC-Client zur AUT her. Im Capture-Modus kann man die Skripte, die zur Automatisierung benötigt werden, aufzeichnen. Man markiert man den repräsentativen Bildausschnitt, zum Beispiel einen Button, auf den man einen Mausklick ausführen will. Nun wird dieser Bildausschnitt mit einem passenden Namen in dem Projekt abgespeichert und ist Eggplant somit bekannt. Über die Eggplant Menüleiste in dem VNC-Client kann man einen Mausklick auf das markierte Objekt ausführen. Dabei wird im Skript folgende Zeile erzeugt:

click(„Objektname“)

Bei einer erneuten Ausführung des Skripts wird dieses Element nun anhand des Bildes wiedergefunden, auch wenn es sich an einer anderen Position im Bild befindet. Die Skripte sind somit robust gegenüber Positionsänderungen von Objekten.

5.5.2 Skriptsprache

Als Skriptsprache wird das von Redstone Software selbst entwickelte Sense Talk verwendet. Bei Sense Talk wird Wert darauf gelegt, dass die Skriptsprache schnell erlernbar ist. Dafür sollte die Syntax intuitiv sein und häufig verwendete Funktionalitäten beherrschen. Sense Talk ist eine interpretierte, objektorientierte Highlevel Sprache, deren Syntax sich stark an der englischen Sprache orientiert. So sind solche Konstrukte möglich:

put **five** is less than **two** times **three**

“Put” ist der Sense Talk Befehl für eine Bildschirmausgabe. Somit erzeugt die obenstehende Zeile die Ausgabe „true“. Natürlich müssen Zahlen nicht ausgeschrieben werden, dieses Beispiel soll nur verdeutlichen, wie stark der Bezug zur gewöhnlichen Sprache in Sense Talk ist.

Hier ein weiteres Beispiel, das auch die Objektorientierung verdeutlich, aus der Sense Talk Dokumentation:

```
put (name:"Michael") into mike -- create an object
put a new object into property cat of mike -- create a nested object
put "Fido" into the name of mike's cat
put mike's cat's name -- Fido
put mike.name – Michael
```

Hier wird der Befehl “put” mit “into” verwendet. In diesem Fall wird ein Wert einer Variablen zugewiesen, die dem Schlüsselwort “into” folgt.

5.5.3 Bilderkennung

Für alle entscheidenden Komponenten werden Bilder in dem Testprojekt abgelegt, anhand derer das Tool beim Ausführen der Testskripte die Objekte erkennt und auf dem Bildschirm lokalisiert. Was passiert aber, wenn ein Button den Fokus besitzt? Nun hat das Icon einen kleinen Rahmen, der nicht auf dem Präsenzbild vorhanden ist. Somit weicht die Darstellung des Icons von dem gespeicherten Bild ab. Für diesen und ähnliche Fälle gibt es in Eggplant die Möglichkeit, mehrere Bilder für ein Objekt zu hinterlegen. Nun wird beim Suchen des Objektes eine Übereinstimmung mit einem der Bilder gesucht. Diese Funktionalität kann auch sinnvoll eingesetzt werden, um plattformübergreifende Tests zu erzeugen. Die Darstellung einer Java-Applikation unter Windows und MacOS unterscheidet sich, wobei die Funktion die gleiche bleiben soll. Speichert man nun zu einem Objekt die Darstellung unter Windows und MacOS ab, kann das gleiche Skript für beide Betriebssysteme verwendet werden.

Für den Fall, dass man eine Aktion auf ein veränderbares Objekt ausführen möchte, gibt es die Möglichkeit, einen sogenannten Hotspot zu setzen. Eine solche Aktion könnte eventuell der Klick auf die Adressleiste des Browsers sein. Würde man ein Bild der Adressleiste speichern, müsste ein Bild für jeden möglichen Inhalt vorhanden sein. Dies wäre bei einem Textfeld eine sehr schwierige, wenn nicht sogar unmögliche Aufgabe. Hier kann sich mit folgendem Trick geholfen werden: Man sucht ein Symbol neben der Adressleiste und kann nun in Eggplant angeben, dass der Klick einige Pixel neben dem gefundenen Element ausgeführt werden soll. Dann wäre der Cursor in der Adressleiste positioniert und könnte zum Beispiel die Eingabe einer URL entgegen nehmen.

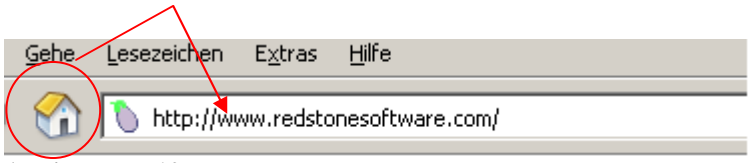


Abbildung 5-10: Eggplant Hotspot setzen

5.5.4 Texterkennung

Es gibt mehrere Situationen, in denen das Erkennen von Texten nötig ist. So kann es notwendig sein, den Inhalt einer Nachricht auszuwerten, um zu erkennen, ob eventuell ein Fehler aufgetreten ist oder ob die richtigen Daten angezeigt werden. Man kann auch eine

Aktion auf einen bestimmten Dialog (z.B.: „Datei“) ausführen. Hier würde man nach dem Wort „Datei“ suchen, um dort die gewünschte Aktion auszuführen.

Natürlich kann man sich in den meisten Fällen mit der einfachen Bilderkennung behelfen. Wenn aber der Entwickler entscheidet, die Schrift zu ändern, ist man gezwungen, alle Skripte zu überarbeiten und alle Bilder neu zu erfassen. Dies wäre sicherlich nicht akzeptabel.

In Eggplant können Schriften dynamisch gerendert werden. Man gibt also in dem Automatisierungsskript lediglich den Text und den Schriftstyle an. Von diesen Styles können verschiedene angelegt und verwaltet werden. Ändert nun der Entwickler die Schriftart, braucht man nur noch den Style anzupassen, und die Skripte sind wieder lauffähig. Im Skript würde eine solche Aktion so aussehen:

Click (text: "Open", textStyle: "WindowsExplorer")

Ein weiterer Vorteil des dynamischen Renderns der Texte ist die Möglichkeit, datengetriebene Tests zu erstellen. Man kann in dem Beispielskript den String „Open“ auch aus einer Textdatei gelesen haben, die als Grundlage für den datengetriebenen Test dient. Normalerweise wird es sich bei einer solchen Datei um eine Tabelle handeln, deren Datensätze in einer Schleife abgearbeitet werden.

5.5.5 Zusammenfassung

Die Erkennung von Steuerelementen anhand von Bildern kann man als Workaround betrachten für den Fall, dass man nicht in der Lage ist, auf die innere Repräsentation dieser Elemente zuzugreifen. Eine weitere Alternative ist das Speichern von Bildschirmkoordinaten, um Aktionen auszuführen. Allerdings ist ein solches Skript nicht sonderlich robust, da es scheitert, sobald ein Element verschoben wird.

Das Anlegen der Skripte in Eggplant ist umständlicher und wohl auch fehleranfälliger als bei vergleichbaren Tools wie QuickTest Professional. Beim Capture and Replay kann man die Elemente nicht einfach nur anklicken, sondern man muss immer einen repräsentativen Ausschnitt der Anzeige als Bild speichern. Gegebenenfalls müssen für ein Objekt sogar mehrere Bilder erfasst werden. Auch ist es nicht möglich, einfach die Texteigenschaft eines Steuerelements auszulesen und auszuwerten. Man muss für den Vergleich immer denselben Schriftstil verwenden, sonst gibt es fälschlicherweise Abweichungen.

Dies klingt alles mehr nach Nachteilen als nach Vorteilen, aber man darf nicht vergessen, dass dieses Verfahren eventuell die einzige Möglichkeit ist, mit einer AUT zu arbeiten, auf deren interne Repräsentation man nicht zugreifen kann. Und in diesem Bereich kann man mit Eggplant fast genau so arbeiten wie mit Quicktest und einer Standard Windowsanwendung. Man ist plattformunabhängig und kann über mehrere VNC-Verbindungen sogar verschiedene AUTs parallel testen. Zum Beispiel ist dies bei einer verteilten Client/Server Anwendung hilfreich. Man könnte über den Client Daten an den Server senden lassen und im nächsten Schritt am Server überprüfen, ob diese korrekt angekommen und verarbeitet worden sind. In diesem Fall sind die Client- und die Serveranwendung jeweils in einer eigenen VNC-Session geöffnet.

Leider fehlt eine direkte Unterstützung des HP Quality Centers, wie sie bei QuickTest Professional verfügbar ist. Da alle Daten von Eggplant als Text bzw. Tiff vorhanden sind und keinerlei Daten in proprietären Formaten verwendet werden, ist es sicher möglich, eine Integration im Quality Center als Workaround selbst zu konstruieren. Eggplant kann als Konsolenanwendung ausgeführt werden und ließe sich so mit einem entsprechenden Startskript über das Quality Center starten. Dies wurde allerdings nicht ausprobiert und basiert lediglich auf den Aussagen des Technikers von Redstone Software, der die Live-Demo durchgeführt hat.

Alle Erkenntnisse basieren zurzeit auf Dokumentationen und einer Live-Demo sowie zwei einstündigen Telefonkonferenzen. Wie leistungsfähig das Tool in der Praxis ist, wird sich erst im Rahmen einer Teststellung evaluieren lassen.

5.6 Froglogic Squish

Bei Squish von der Hamburger Firma Froglogic handelt es sich um Tool für automatisierte GUI-Tests mit Applikationen, die auf dem QT-Toolkit basieren. Squish hat die Möglichkeit, die innere Repräsentation von QT-Elementen in einer AUT zu ermitteln. Voraussetzung ist, dass der Zugriff auf eine shared QT-Library, die auch die AUT verwendet, gewährleistet ist. Squish wird also auf dem Server selbst installiert, der die AUT bereit stellt. Es kann zum Beispiel über eine XWindow-Session von einem Windowsrechner aus bedient werden. Für Squish existiert ein Plug-In, welches eine Integration mit dem HP Qualitiy Center ermöglicht. Somit können Testfälle in der Datenbank des Quality Centers abgelegt werden. Außerdem können die Tests über das Quality Center ausgeführt und verwaltet werden.

5.6.1 Skripterstellung

Die Erstellung des Automatisierungsskripts erfolgt in Squish-IDE im Capture and Replay Verfahren. Die AUT wird von Squish gestartet, und die Benutzeraktionen mit dieser werden aufgezeichnet, daraus ergibt sich ein Skript in der zuvor gewählten Sprache. Je nachdem, welche Skriptsprachen auf dem Rechner unterstützt werden, stehen JavaScript, Tcl, Python und Perl zur Verfügung. Mit diesem aufgezeichneten Skript lassen sich die Interaktionen automatisch wiederholen.

5.6.2 Verifikation der Ergebnisse

Zur automatisierten Überprüfung der Ergebnisse können sogenannte „Verification Points“ in das Skript eingepflegt werden. Es wird ein Breakpoint an der Stelle im Skript gesetzt, an welcher der „Verification Point“ eingefügt werden soll. Nun lässt man das Skript bis zu diesem Breakpoint laufen und fügt den „Verification Point“ ein. Dies passiert per Knopfdruck in der Squish-IDE. Hier hat man nun den „Squish Spy“ zur Unterstützung. Der „Spy“ erhält eine Liste alle Steuerelemente, die in der ausgewählten Maske der AUT sichtbar sind. Nun kann man mit Anhakfeldern die Eigenschaften verschiedener Elemente markieren, die für die Überprüfung relevant sind. Die aktuellen Werte werden als Referenzwerte übernommen. Bei jeder weiteren Ausführung des Skripts wird nun überprüft, ob alle angehakten Felder die erwarteten Werte enthalten. Werte der Elemente können Inhalte von Textfeldern sein, aber

auch Anzahl von Knoten in einer Liste oder andere Eigenschaften, die das jeweilige Element besitzt.

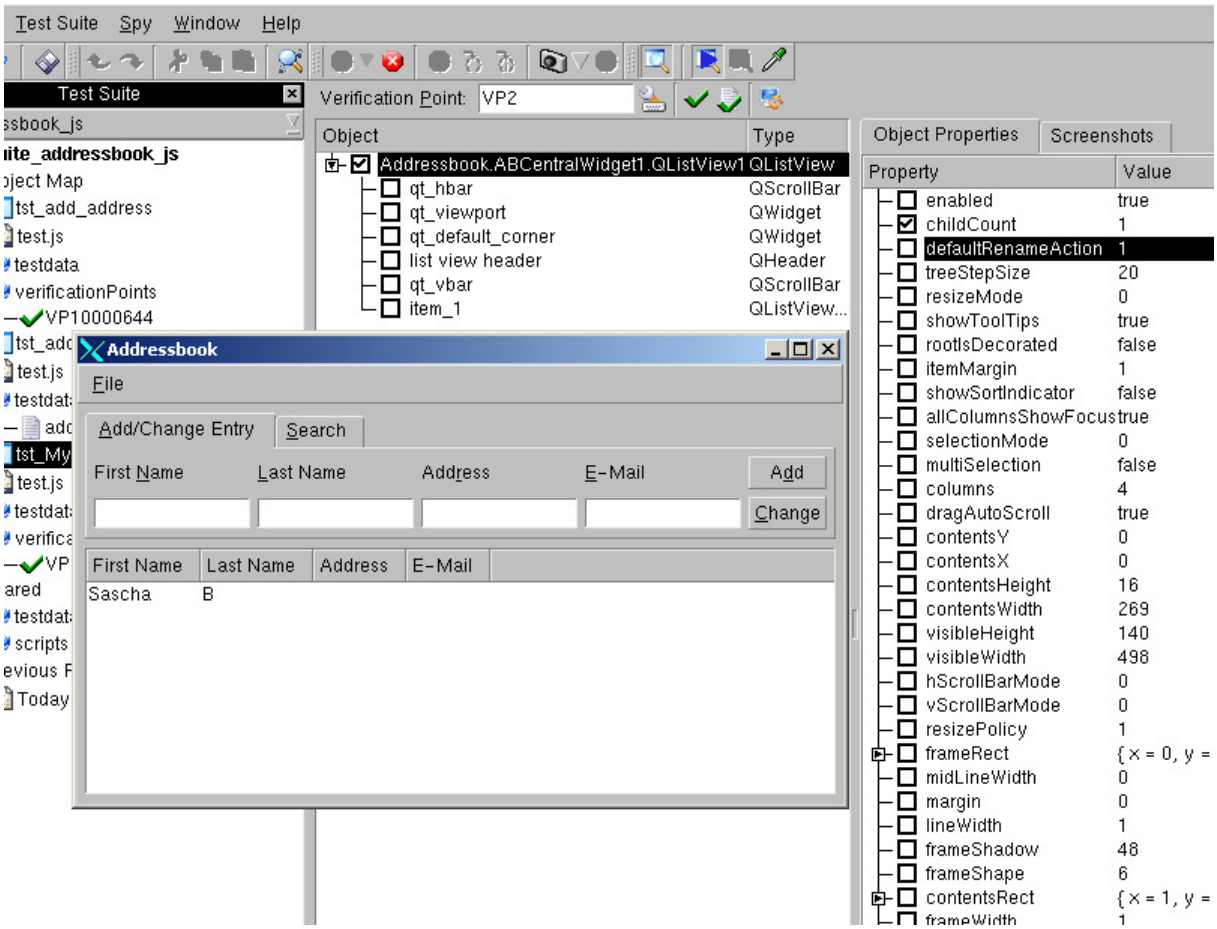


Abbildung 5-11: Squish Adressbuch Beispiel

Es besteht auch die Möglichkeit, ohne Unterstützung des „Spy“ „Verification Points“ zu erstellen. Möchte man zum Beispiel sicherstellen, dass eine bestimmte Datei erzeugt wurde, reicht im Skript zur Überprüfung die Zeile *test.verify(QFile.Exists(„Name“))* aus. Hier kann man alle Möglichkeiten, die die jeweilige Skriptsprache bietet, verwenden.

Nach der Ausführung des Testskripts werden im Log-Fenster im unteren Teil der Squish-GUI die Ergebnisse angezeigt. Dort wird für jeden „Verification Point“ angezeigt, ob er erfolgreich überprüft werden konnte oder nicht. Auch die Abweichungen von den erwarteten Werten werden hier angezeigt.

Neben der Verifikation über die Eigenschaften der Steuerelemente bietet Squish auch eine Überprüfung anhand von Screenshots.

Es gibt unterschiedliche Möglichkeiten, die Skripte robust zu machen. Das heißt, dass die Skripte nicht durch unrelevante Abweichungen einen Fehlerzustand melden. Wenn bei einem Test eines FTP-Clients die Netzwerkverbindung etwas langsam ist und in einem Fenster nicht sofort alle erwarteten Dateien angezeigt werden, könnte dies dazu führen, dass ein Fehlerzustand gemeldet wird, obwohl wenige Sekunden später vielleicht alle Dateien ordnungsgemäß angezeigt werden. Hier hat man die Möglichkeit, ein WaitFor-Statement in

das Skript einzubauen. Dieses Statement kann verwendet werden, um darauf zu warten, dass eine bestimmte Anzahl an Elementen in einer Liste angezeigt werden – in diesem Beispiel also die Anzahl der Dateien in der Browserliste.

Ein weiteres Problem für die Robustheit eines Skriptes können Anzeigen mit dem aktuellem Datum oder der Uhrzeit sein, da diese garantiert von der Anzeige bei der Erstellung des Skriptes abweichen werden. Im Skript können solche Prüfungen mit Hilfe von Wildcards, also „*“ oder „?“, verbessert werden.

5.6.3 Zusammenfassung

Zum Testen von Anwendungen, die auf dem QT-Toolkit von Trolltech basieren, scheint Squish die erste Wahl zu sein. Das bedeutet auf der anderen Seite natürlich, dass man mit Squish nur QT-Anwendungen testen kann. Es gibt von Squish auch Versionen für Tests von Web-Applikation und Javaprogrammen und mehr. Trotzdem verlangt eine anders geartete AUT auch den Einsatz eines anderen Tools, sei es eine andere Squish-Version oder ein ganz anderes Programm.

Das Erstellen der Skripte in Squish erscheint intuitiv durch das Capture and Replay Verfahren. Die „Verification Points“ können komfortabel eingepflegt werden und bieten viele Möglichkeiten. Die Auswahl verschiedener Skriptsprachen (JavaScript, Tcl, Python, Perl) bietet genügend Flexibilität, um in den meisten Umgebungen zu funktionieren. Es ist nicht nötig, eine proprietäre Skriptsprache zu erlernen.

Verschiedene Addons, die unter anderem die Integration in das HP Quality Center ermöglichen, helfen dabei, Squish einfach und schnell in die Testabläufe einzubinden und effektiv zu nutzen.

5.7 Ergebnis der Toolauswahl

Die Auswahlkriterien aus dem Kapitel 5.1.2 Auswahlkriterien werden von den drei Tools folgendermaßen erfüllt:

Froglogic Squish

ID	Beschreibung	Erfüllt
1	Integration in Quality Center ist mit einem Plug-In möglich	ja
2	Für QT-Anwendungen konzipiert	ja
3	Eine andere Version, für die eine weitere Lizenz benötigt wird, kann möglicherweise Windows-Objekte erkennen	nein
4	Es werden gängige Skriptsprachen wie JavaScript verwendet	ja
5	Es kann auf Datenbanksysteme zugegriffen werden	ja
6	Lasttests können ausgeführt werden	ja
7	Läuft unter Solaris und lässt sich per Host-Emulation von Windows aus bedienen	ja
8	ca. 2.400 EUR	ja

HP QuickTest Professional

ID	Beschreibung	Erfüllt
1	Integration in Quality Center ist möglich	ja
2	Keine Erkennung von QT-Objekten	nein
3	Für Windows-Anwendungen konzipiert	ja
4	Visual Basic Script ist eine gängige und einfach zu erlernende Skriptsprache	ja
5	Es kann auf Datenbanksysteme zugegriffen werden	ja
6	Lasttests können nicht ausgeführt werden	nein
7	Läuft unter Windows	ja
8	ca. 3.000 EUR	ja

Redstonesoftware Eggplant

ID	Beschreibung	Erfüllt
1	Integration in Quality Center ist nicht möglich	ja
2	Erkennung von QT-Objekten durch Bilderkennung	ja
3	Erkennung von Windows-Objekten durch Bilderkennung	ja
4	Eigene Skriptsprache Sense Talk. Intuitiv	ja
5	Datenbankzugriff ist in Planung	nein
6	Lasttests können nicht ausgeführt werden	nein
7	Benötigt Mac OS, kann per VNC von Windows bedient werden	nein
8	ca. 3.750 EUR	ja

Der Wunsch, alle Tests mit einem Tool ausführen zu können, ist nur mit Eggplant zu realisieren. Leider ist die Implementation der Tests durch die Bilderkennung eher umständlich, auch wenn dadurch die Abhängigkeit von der Objekterkennung umgangen wird und somit über Plattformgrenzen hinaus gearbeitet werden kann. Außerdem fehlt eine Integrationsmöglichkeit zum HP Quality Center.

Die Variante, mit zwei Tools zu arbeiten, also HP QuickTest Professional für COIN und Squish für TOPX zu verwenden, bietet erhebliche Vorteile. Beide Tools arbeiten mit dem HP Quality Center zusammen, und die Realisierung ist erheblich günstiger. Für QuickTest Professional müssen keine Lizenzen angeschafft werden, Squish hingegen ist erheblich preisgünstiger als Eggplant. Für den Einsatz von Eggplant würden zusätzlich noch die Kosten für die Anschaffung von Mac OS anfallen.

6 Automatisierung der Tests für die COIN-Migration

In diesem Kapitel wird beschrieben, wie der Testprozess in diesem Migrationsprojekt mit automatisieren Tests verbessert werden konnte. In diesem recht speziellen Projekt sollte das textbasierte, in COBOL entwickelte COIN durch eine neue Software abgelöst werden. Das Besondere liegt hierbei darin, dass die neue Software genau die gleichen Anforderungen hat wie das Altsystem. Es soll lediglich eine moderne Architektur verwendet werden. Das neue System soll durch einen stark automatisierten Konvertierungsprozess entstehen und nicht durch eine manuelle Neuimplementierung. Im Test soll geprüft werden, ob das Neusystem tatsächlich die gleichen Anforderungen wie das Altsystem erfüllt.

6.1 Testkonzept

Dieses Kapitel behandelt nicht das gesamte Testkonzept der COIN-Migration, sondern nur die Aspekte, die mit einer Automatisierung im Zusammenhang mit dieser Arbeit betrachtet wurden.

Hier eine Zusammenfassung der Ausgangssituation:

- Es sollen zwei Programme auf Gleichheit in verschiedenen Punkten überprüft werden.
- Die Bedienung der Programme ist durch das Aufrufen verschiedener Funktionen mit einem Schlüssel nicht sehr umfangreich (im Vergleich zu einer GUI mit Mausbedienung und mehreren Fenstern).
- Im Rahmen dieser Arbeit soll gezeigt werden, wie sich beide Programme mit QuickTest Professional automatisieren lassen.
- Es sollen viele Dialoge und Funktionen überprüft werden.

Bevor man mit einer Automatisierung beginnt, sollte man sich die Fragen stellen, ob sich eine Automatisierung überhaupt lohnt: Ist die automatische Ausführung der Tests schneller und/oder gründlicher als die manuelle Ausführung? Bei der COIN-Migration sollen Dialoge Zeichen für Zeichen miteinander verglichen werden, dies erscheint als eine typische Aufgabe für einen Rechner. Für einen automatisierten Vergleich muss also eine Möglichkeit gefunden werden, diese Masken rechnergestützt zu vergleichen. Die Idee ist daher, dass der gleiche Funktionsaufruf mit dem gleichen Datenbestand die gleiche Ausgabe erzeugt. Dafür sind zwei Voraussetzungen nötig:

1. Automatisiertes Aufrufen der Funktionen
2. Automatisiertes Überprüfen der Maskeninhalte

6.2 Automatisiertes Aufrufen der Funktionen

Für den automatischen Aufruf der Funktionen wird HP QuickTest Professional verwendet. Hiermit ist es möglich, Tastatureingaben an das Exceed-Fenster zu schicken. Mit Hilfe des Capture and Replay Modus wurde die Eingabe einer Funktion mit Schlüssel aufgezeichnet. Die anfängliche Positionierung des Eingabecursors erfolgt per Mausklick. Aufgrund der fehlenden Erkennung von Objekten werden für diesen Klick nur die Mauskoordinaten erfasst.

Da sich das Eingabefeld für die Funktion in jeder Maske an der gleichen Stelle befindet, entsteht daraus kein Problem für den verlässlichen Ablauf des Skriptes. QuickTest Professional hat diesen VBScript Code erzeugt:

```
'Coin Fenster in den Vordergrund bringen
Window("ExceedWindow").Activate
'Cursor per Mausklick in das Funktionsfeld bringen
Window("ExceedWindow").Click 173,498
‘Funktion eingeben
Window("ExceedWindow").Type “362AZ”
‘in das nächste Eingabefeld springen
Window("ExceedWindow").Type micTab
‘Schlüssel eingeben
Window("ExceedWindow").Type “tghu9507987”
'Mit "Return" Funktion ausführen
Window("ExceedWindow").Type  micReturn
```

Ohne die letzte Zeile sieht das COIN-Fenster nach dem Auflauf des Skripts wie folgt aus:

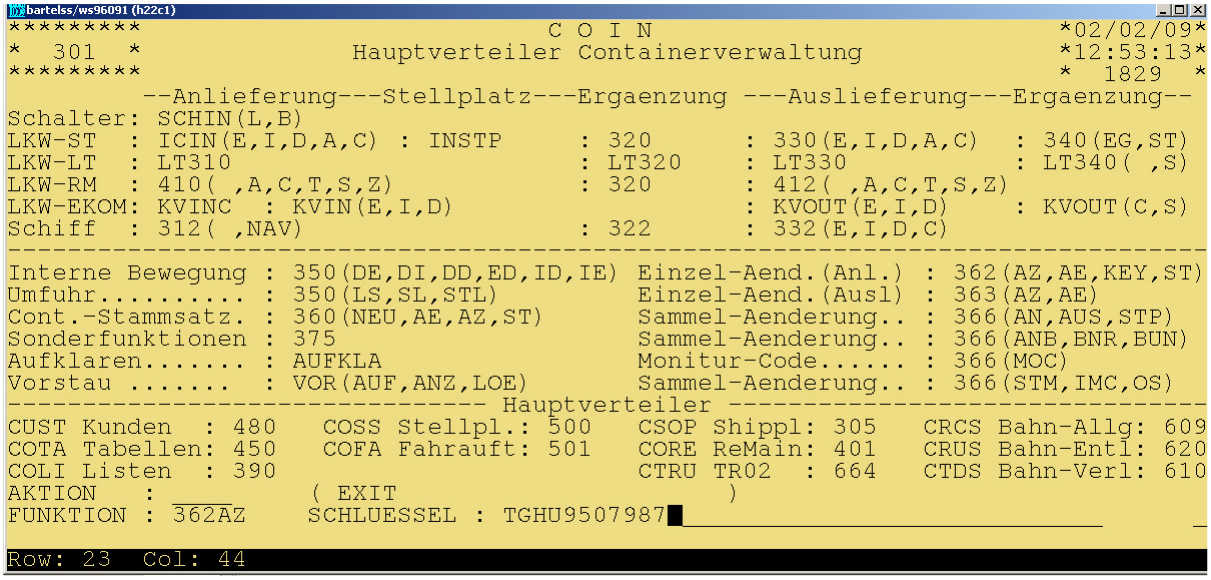

bartelss/ws96091 (h22c1)
******** C O I N *02/02/09*
* 301 * Hauptverteiler Containerverwaltung *12:53:13*
******** * 1829 *
--Anlieferung---Stellplatz---Ergaenzung ---Auslieferung---Ergaenzung--
Schalter: SCHIN(L,B)
LKW-ST : ICIN(E,I,D,A,C) : INSTP : 320 : 330(E,I,D,A,C) : 340(EG,ST)
LKW-LT : LT310 : LT320 : LT330 : LT340(,S)
LKW-RM : 410(,A,C,T,S,Z) : 320 : 412(,A,C,T,S,Z)
LKW-EKOM: KVINC : KVIN(E,I,D) : KVOUT(E,I,D) : KVOUT(C,S)
Schiff : 312(,NAV) : 322 : 332(E,I,D,C)
Interne Bewegung : 350(DE,DI,DD,ED,ID,IE) Einzel-Aend.(Anl.) : 362(AZ,AE,KEY,ST)
Umfuhr..... : 350(LS,SL,STL) Einzel-Aend.(Ausl) : 363(AZ,AE)
Cont.-Stammsatz. : 360(NEU,AE,AZ,ST) Sammel-Aenderung.. : 366(AN,AUS,STP)
Sonderfunktionen : 375 Sammel-Aenderung.. : 366(ANB,BNR,BUN)
Aufklaren..... : AUFKLA Monitur-Code..... : 366(MOC)
Vorstau ..... : VOR(AUF,ANZ,LOE) Sammel-Aenderung.. : 366(STM,IMC,OS)
Hauptverteiler
CUST Kunden : 480 COSS Stellpl.: 500 CSOP Shippl: 305 CRCS Bahn-Allg: 609
COTA Tabellen: 450 COFA Fahrauft: 501 CORE ReMain: 401 CRUS Bahn-Entl: 620
COLI Listen : 390 CTRU TR02 : 664 CTDS Bahn-Verl: 610
AKTION : ____ (EXIT)
FUNKTION : 362AZ SCHLUESSEL : TGHU9507987
Row: 23 Col: 44

Abbildung 6-1: COIN-Fenster mit Funktionsaufruf

Die Funktion 362AZ soll eine Maske aufrufen, die Informationen über einen Container darstellt. „TGHU9507987“ ist eine gültige Containernummer. Durch die Returntaste wird die Funktion ausgeführt, und es wird ein neuer Dialog angezeigt, der die Daten zu dem Container anzeigt.

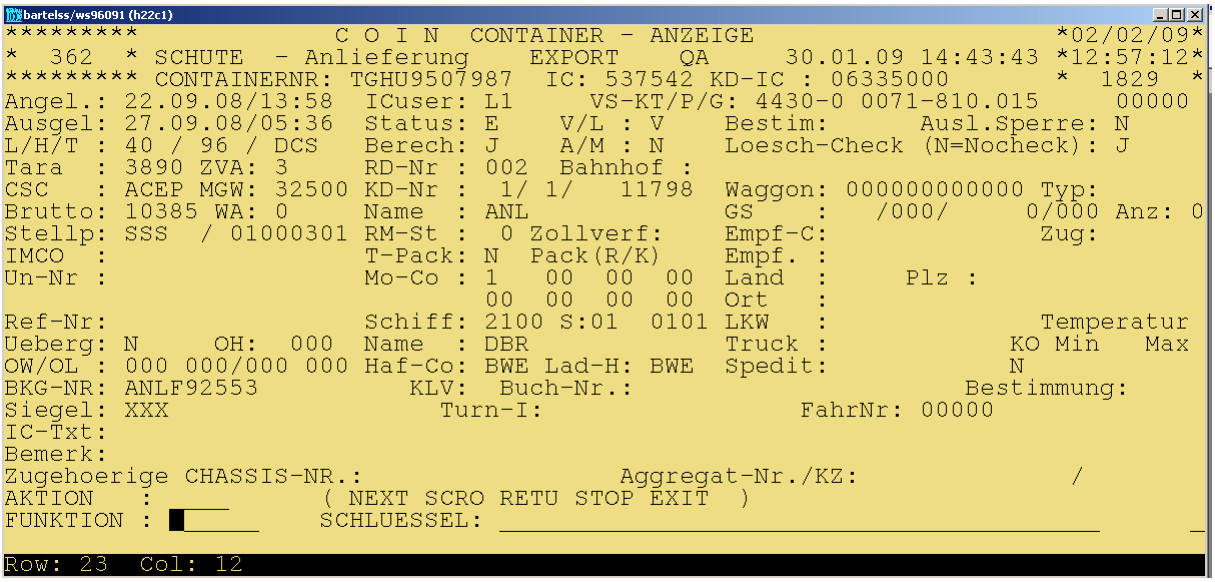
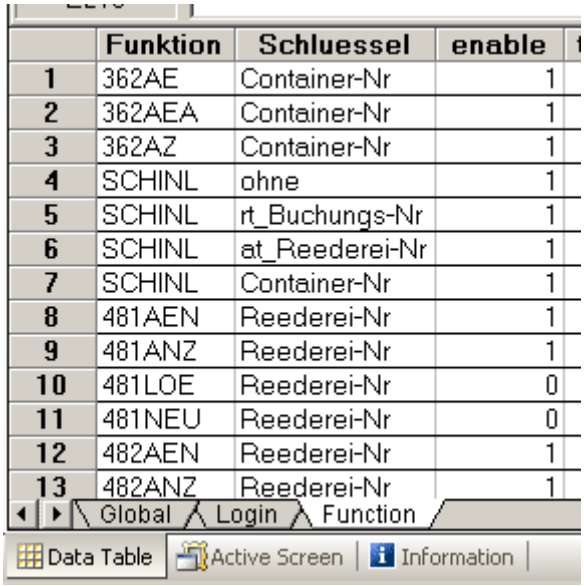

bartelss/ws96091 (h22c1)
********* C O I N CONTAINER - ANZEIGE *02/02/09*
* 362 * SCHUTE - Anlieferung EXPORT QA 30.01.09 14:43:43 *12:57:12*
********* CONTAINERNR: TGHU9507987 IC: 537542 KD-IC : 06335000 * 1829 *
Angel.: 22.09.08/13:58 ICuser: L1 VS-KT/P/G: 4430-0 0071-810.015 00000
Ausgel: 27.09.08/05:36 Status: E V/L : V Bestim: Ausl.Sperre: N
L/H/T : 40 / 96 / DCS Berech: J A/M : N Loesch-Check (N=Nocheck): J
Tara : 3890 ZVA: 3 RD-Nr : 002 Bahnhof :
CSC : ACEP MGW: 32500 KD-Nr : 1/ 1/ 11798 Waggon: 000000000000 Typ:
Brutto: 10385 WA: 0 Name : ANL GS : /000/ 0/000 Anz: 0
Stellp: SSS / 01000301 RM-St : 0 Zollverf: Empf-C: Zug:
IMCO : T-Pack: N Pack(R/K) Empf. :
Un-Nr : Mo-Co : 1 00 00 00 Land : Plz :
00 00 00 00 Ort :
Ref-Nr: Schiff: 2100 S:01 0101 LKW : Temperatur
Ueberg: N OH: 000 Name : DBR Truck : KO Min Max
OW/OL : 000 000/000 000 Haf-Co: BWE Lad-H: BWE Spedit: N
BKG-NR: ANLF92553 KLV: Buch-Nr.: Bestimmung:
Siegel: XXX Turn-I: FahrNr: 00000
IC-Txt:
Bemerk:
Zugehoerige CHASSIS-NR.: Aggregat-Nr./KZ: /
AKTION : (NEXT SCRO RETU STOP EXIT)
FUNKTION : SCHLUESSEL:
Row: 23 Col: 12

Abbildung 6-2: COIN-Maske 362 – Containeranzeige

Nachdem der Aufruf einer Maske automatisiert wurde, ist das nächste Ziel, den Aufruf zu parametrisieren und somit beliebige Kombinationen von Funktionen und Schlüsseln aufrufen zu können. Zur Unterstützung eines solchen datengetriebenen Tests bietet sich die Verwendung der „Data Table“ von QuickTest Professional an. (Vgl. das Kapitel 5.3.3 Datengetriebener Test)

Es werden Parameter für den Funktionsnamen und für den Schlüsseltyp benötigt. In der Data Table sieht dies so aus:



	Funktion	Schluessel	enable
1	362AE	Container-Nr	1
2	362AEA	Container-Nr	1
3	362AZ	Container-Nr	1
4	SCHINL	ohne	1
5	SCHINL	rt_Buchungs-Nr	1
6	SCHINL	at_Reederei-Nr	1
7	SCHINL	Container-Nr	1
8	481AEN	Reederei-Nr	1
9	481ANZ	Reederei-Nr	1
10	481LOE	Reederei-Nr	0
11	481NEU	Reederei-Nr	0
12	482AEN	Reederei-Nr	1
13	482ANZ	Reederei-Nr	1

Abbildung 6-3: Data Table in QuickTest Professional

Die Spaltenüberschriften sind der Parametername, der später in dem Automatisierungsskript zur Verfügung steht. Die Zelleninhalte sind die dazugehörigen Werte. Jede Zeile stellt einen neuen Testfall dar. Die Spalte „enable“ wurde eingeführt, um von Fall zu Fall entscheiden zu können, ob die Funktion tatsächlich ausgeführt werden soll. Als Schlüssel sind in dieser Tabelle weitere Parameternamen eingetragen. Somit wird es ermöglicht, für alle Tests einen globalen Wert für jeden Schlüsseltypen festzulegen. Dadurch wird die Wartbarkeit der Tests

erhöht, da nur an einer zentralen Stelle neue Daten eingetragen werden müssen, wenn die Tests mit einem anderen Datenbestand ausgeführt werden sollen.

Diese zentrale Stelle befindet sich hinter dem Reiter „Global“ der Data Table. Die dort eingetragenen Werte sind in allen Testfällen gültig. In QuickTest Professional wird von „Actions“ gesprochen. In diesem Projekt gibt es also eine Action „Login“ und eine Action „Function“. In der Action „Login“ kann nur auf Werte zugegriffen werden, die in den Datenblättern „Login“ und „Global“ festgelegt wurden. Die Action „Login“ meldet sich lediglich an COIN an, damit dann die Tests der Funktionen durchgeführt werden können.

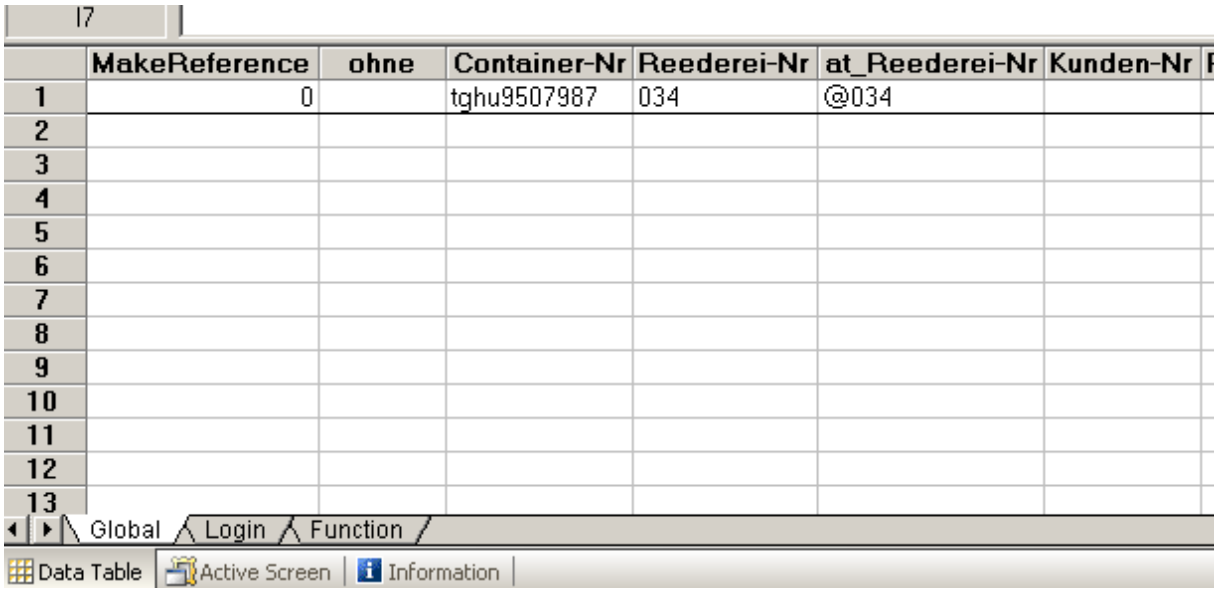


Abbildung 6-4: QuickTest Professional Data Table "Global"

Mit dieser Konfiguration wird jede Funktion, die eine Containernummer benötigt, mit der Nummer „tghu9507987“ aufgerufen, und jede Funktion, die eine Reedereinummer als Schlüssel erwartet, mit „034“. Es gibt Funktionen, die verschiedene Schlüsseltypen akzeptieren. Damit von der Funktion erkannt werden kann, um welchen Typ es sich handelt, werden diese manchmal mit Präfixen versehen. Dies ist hier an dem Schlüssel „at_Reederei-Nr“ zu erkennen. Für jeden Tupel aus Funktion und Schlüsseltypen gibt es einen Eintrag im Datenblatt „Function“, also auch einen Testfall pro Tupel. Auf die Bedeutung der Spalte „MakeReference“ wird im folgenden Kapitel eingegangen.

Das VBSkript muss nun wie folgt angepasst werden:

```
If  DataTable("enable", dtLocalSheet) = 1 Then
'enable kennzeichnet, ob dieser Testfall tatsächlich ausgeführt werden soll
        'Coin Fenster in den Vordergrund bringen
        Window("ExceedWindow").Activate
        'Cursor per Mausklick in das Funktionsfeld bringen
        Window("ExceedWindow").Click 173,498
        'Funktion ausführen
        Window("ExceedWindow").Type DataTable("Funktion", dtLocalSheet)

        ‘Schlüssel eingeben
        Window("ExceedWindow").Type
                DataTable( DataTable("Schluessel",dtLocalSheet),dtGlobal)
```

```
        'Mit "Return" Funktion ausführen
        Window("ExceedWindow").Type  micReturn
End If
```

Der Aufruf DataTable (Parameter, Datenblatt) gibt den Wert des jeweiligen Parameters aus dem angegebenen Datenblatt zurück. Bei dem verschachtelten Aufruf *DataTable(DataTable("Schluessel",dtLocalSheet),dtGlobal)* wird zunächst der Schlüsseltyp aus dem „Local Sheet“, dem Datenblatt der aktuellen Action – hier „Function“– ermittelt und an das globale Datenblatt geleitet, um den tatsächlichen Wert zu erhalten.

Als nächstes muss noch in dem Menü „Test Settings“ bei „Data Table Iterations“ die Option „Run on all rows“ gewählt werden. Dies bedeutet, dass für jede Zeile in der Data Table der Test einmal durchlaufen werden soll. In diesem Fall hat das globale Datenblatt eine Zeile und das Datenblatt für die Action „Function“ ca. 500 Zeilen. Also wird beim Durchlauf des ganzen Tests die Action „Function“ ca. 500-mal ausgeführt. Damit ist die Automatisierung der Funktionsaufrufe von technischer Seite her abgeschlossen.

Wie gut diese Automatisierung funktioniert, hängt in diesem Fall von der Qualität der Daten aus der Data Table ab. Die Grundlage für diese Daten bildet eine Excel-Tabelle, in der die Funktionen von den Entwicklern des Altsystems dokumentiert wurden. Diese Dokumentation wurde für das Migrationsprojekt angelegt und nicht von Beginn der COIN-Entwicklung an gepflegt. Aufgrund der großen Anzahl an Funktionen und dem somit verbundenen Aufwand der Erstellung dieser Dokumentation können Fehler und Unvollständigkeiten auftreten. Diese Daten wurden parallel in dem System „Adonis“, einer Software zur Dokumentation von Geschäftsprozessen, hinterlegt. Von dort ist es möglich, die Daten in eine Excel-Tabelle zu exportieren. Somit kann nach einer Fehlerkorrektur eine aktuelle Exceltabelle erstellt werden, die mit wenig Aufwand für den Test aufbereitet werden kann.

6.3 Automatisiertes Überprüfen der Maskeninhalte

Um die Inhalte der Masken der beiden Systeme zu vergleichen, gibt es im Prinzip drei Lösungsansätze. Erstens: Es können beide Systeme parallel ausgeführt werden und die jeweiligen Masken direkt vergleichen werden. Zweitens: Ein System wird ausgeführt, und die Ausgaben werden gespeichert. Anschließend wird das zweite System ausgeführt, auch hier werden die Ausgaben gespeichert. Danach werden die gespeicherten Daten verglichen. Drittens: Zuerst werden (wie bei Variante zwei) mit dem ersten System Referenzdaten erzeugt. Aber bei der Ausführung des zweiten Systems werden die Ausgaben sofort mit den Referenzen verglichen und nicht abgespeichert. Nur im Fehlerfall müssen die Ausgaben des zweiten Systems gespeichert werden, um die Abweichungen nachvollziehen zu können.

Hier fiel die Entscheidung für die dritte Variante. Somit können bereits Referenzdaten mit dem Altsystem erstellt werden, bevor das Neusystem zur Verfügung steht. Da nicht alle Ausgaben des Neusystems beim Test gespeichert werden, wird das Anlegen überflüssiger Daten vermieden. Zu den Referenzdaten werden nur Daten angelegt, die Fehler im Neusystem beschreiben. Die gespeicherten Daten sind Textdateien, die den jeweiligen Maskeninhalt repräsentieren.

6.3.1 Erstellung von Referenzdaten

Die Erstellung der Referenzdaten wird mit einer im Rahmen dieser Arbeit entwickelten VBSkript Bibliothek vorgenommen. Diese Bibliothek kann in mehreren QuickTest Professional Projekten referenziert werden und enthält weitere Funktionen zum Vergleich und zur Aufbereitung der Testdaten. Die Reihenfolge und die Art der Funktionen und Parameter, die aufgerufen werden, wird über die Data Table gesteuert.

Mit dem Parameter „MakeReferences“ in dem globalen Datenblatt kann entschieden werden, ob die Ausgaben gespeichert werden sollen oder ob mit bereits gespeicherten Daten verglichen werden soll. Die gespeicherten Ausgaben werden Referenzdaten genannt. Der Test soll so aussehen, dass mit dem Altsystem Referenzdaten erstellt werden und diese dann mit den Ausgaben des Neusystems verglichen werden. Als Format für die Referenzdaten wurden Textdateien gewählt. Diese lassen sich einfach erzeugen und jederzeit per Editor einsehen. Das zu erwartende Datenaufkommen ist nicht so hoch, als dass es sich lohnen würde, eine Datenbank zu verwenden. Auch werden keine komplizierten Suchabfragen benötigt. Mit dem VBSkript Befehl *Window("ExceedWindow").GetVisibleText* wird der aktuell angezeigte Maskeninhalt als String zurückgegeben. Bei der Maske 362 sieht der Rückgabewert folgendermaßen aus:

```
bartelss/ws96091 (h22c1)
*********             C O I N  CONTAINER - ANZEIGE                    *02/02/09*
*  362  * SCHUTE  - Anlieferung    EXPORT    QA     30.01.09 14:43:43 *15:43:10*
********* CONTAINERNR: TGHU9507987  IC: 537542 KD-IC : 06335000       *  1832  *
Angel.: 22.09.08/13:58  ICuser: L1     VS-KT/P/G: 4430-0 0071-810.015     00000
Ausgel: 27.09.08/05:36  Status: E    V/L : V    Bestim:      Ausl.Sperre: N
L/H/T : 40 / 96 / DCS   Berech: J    A/M : N    Loesch-Check (N=Nocheck): J
Tara  : 3890 ZVA: 3     RD-Nr : 002  Bahnhof :
CSC   : ACEP MGW: 32500 KD-Nr :  1/ 1/   11798  Waggon: 000000000000 Typ:
Brutto: 10385 WA: 0     Name  : ANL            GS    :   /000/     0/000 Anz: 0
Stellp: SSS  / 01000301 RM-St :  0 Zollverf:    Empf-C:              Zug:
IMCO  :                 T-Pack: N  Pack(R/K)    Empf. :
Un-Nr :                 Mo-Co : 1   00  00  00  Land  :     Plz :
                                  00  00  00  00  Ort   :
Ref-Nr:                 Schiff: 2100 S:01  0101 LKW   :              Temperatur
Ueberg: N     OH:  000  Name  : DBR            Truck :            KO Min   Max
OW/OL : 000 000/000 000 Haf-Co: BWE Lad-H: BWE  Spedit:            N
BKG-NR: ANLF92553          KLV:  Buch-Nr.:                      Bestimmung:
Siegel: XXX                  Turn-I:                FahrNr: 00000
IC-Txt:
Bemerk:
Zugehoerige CHASSIS-NR.:                Aggregat-Nr./KZ:             /
AKTION   : ____      ( NEXT SCRO RETU STOP EXIT  )
FUNKTION : ______    SCHLUESSEL: ____________________________________________      _

Row: 23  Col: 12
```

In dieser Maske sind noch einige Daten vorhanden, die für einen automatisierten Vergleich problematisch sind. Die erste Zeile besteht aus dem Windows-Anmeldenamen und der Rechnernummer. Diese Informationen stammen von dem Exceed-Fenster und werden nicht in COIN angezeigt, sie sind für den Test irrelevant und würden bei einer Ausführung mit einem anderen Benutzernamen und/oder auf einem anderen Rechner zu einer falschen Fehlermeldung führen. Die Angaben in der rechten oben Ecke würden ebenfalls zu einem Fehler führen. Dies sind das aktuelle Datum, die Uhrzeit und eine Session ID. Alle drei Angaben wären als Referenzdaten bei einem späteren Vergleich nicht identisch mit den

Werten der aktuell angezeigten Maske. In der letzten Zeile wird die aktuelle Position des Cursors in der COIN-Maske angezeigt. Diese Angabe könnte unter Umständen sogar hilfreich sein, wird jedoch von dem neuen COIN-System nicht geliefert. Alle diese Daten werden bei den Referenzdaten nicht gespeichert. In einigen Masken werden Datum und Uhrzeit noch mal in der dritten oder vierten Zeile angezeigt, auch diese Daten müssen vor dem Speichern gelöscht, also durch Leerzeichen ersetzt werden.

Nach der Entfernung dieser variablen Werte sieht der Inhalt der Textdatei wie folgt aus.

```
*********             C O I N  CONTAINER - ANZEIGE                    *
*  362  * SCHUTE  - Anlieferung    EXPORT    QA                        *
********* CONTAINERNR: TGHU9507987  IC: 537542 KD-IC : 06335000       *
Angel.: 22.09.08/13:58  ICuser: L1     VS-KT/P/G: 4430-0 0071-810.015     00000
Ausgel: 27.09.08/05:36  Status: E    V/L : V    Bestim:      Ausl.Sperre: N
L/H/T : 40 / 96 / DCS   Berech: J    A/M : N    Loesch-Check (N=Nocheck): J
Tara  : 3890 ZVA: 3     RD-Nr : 002  Bahnhof :
CSC   : ACEP MGW: 32500 KD-Nr :  1/ 1/   11798  Waggon: 000000000000 Typ:
Brutto: 10385 WA: 0     Name  : ANL            GS    :   /000/     0/000 Anz: 0
Stellp: SSS  / 01000301 RM-St :  0 Zollverf:    Empf-C:              Zug:
IMCO  :                 T-Pack: N  Pack(R/K)    Empf. :
Un-Nr :                 Mo-Co : 1   00  00  00  Land  :     Plz :
                                00  00  00  00  Ort   :
Ref-Nr:                 Schiff: 2100 S:01  0101 LKW   :              Temperatur
Ueberg: N     OH:  000  Name  : DBR            Truck :            KO Min   Max
OW/OL : 000 000/000 000 Haf-Co: BWE Lad-H: BWE  Spedit:            N
BKG-NR: ANLF92553          KLV:  Buch-Nr.:                    Bestimmung:
Siegel: XXX                  Turn-I:               FahrNr: 00000
IC-Txt:
Bemerk:
Zugehoerige CHASSIS-NR.:                Aggregat-Nr./KZ:             /
AKTION   : ____       ( NEXT SCRO RETU STOP EXIT  )
FUNKTION : _______    SCHLUESSEL: ____________________________________________      _
```

In diesem Format können die Textdateien als Referenzdaten verwendet werden. Um die Dateien sicher den entsprechenden Testfällen zuordnen zu können, werden Dateinamen mit folgender Syntax erzeugt: Funktionsname_Schlüsseltyp.txt. In diesem Beispiel also „362AZ_Container-Nr.txt“

6.3.2 Vergleich mit Referenzdaten

Nachdem für alle dokumentierten Funktionen Referenzdaten im Altsystem erzeugt wurden, sollen nun die Ausgaben des Neusystems anhand dieser Daten verifiziert werden. Da zum Zeitpunkt der Entwicklung des Komparators das Neusystem noch nicht zur Verfügung stand, wurden zum Überprüfen der Tests die Referenzdaten mit den Ausgaben des Altsystems verglichen. Es wurde also das Programm mit sich selbst überprüft. Dabei war zu erwarten, dass keine Abweichungen auftreten. Im nächsten Schritt wurden die Referenzdaten nachträglich manipuliert, um zu zeigen, dass der Komparator diese Veränderungen zuverlässig aufdeckt.

Die Maskeninhalte des Altsystems werden mit der gleichen Funktion wie die Referenzdaten aufbereitet. Der Vergleich ist einfach zu implementieren, da nur 2 Strings zeichenweise miteinander verglichen werden müssen. Wichtig für die Effizienz des Tests ist eine übersichtlich gestaltete Auswertung der Ergebnisse. Am Ende eines Testdurchlaufs zeigt QuickTest Professional die Ergebnisse im Reporter an. Den Status der einzelnen Testschritte

und die angezeigten Informationen können auch per Quellcode gesetzt werden. Dies ist dann notwendig, wenn nicht die von QuickTest Professional bereit gestellten Checkpoints verwendet werden.

Mit dem Aufruf:

Reporter.ReportEvent *EventStatus*, *ReportStepName*, *Details*

können Einträge im Reporter erzeugt werden. Der Status kann drei mögliche Werte annehmen: Fail, Warning und Pass. In den Details sollten dann die relevanten Daten, die zur Nachvollziehbarkeit des Testergebnisses wichtig sind, angegeben werden. Bei fehlgeschlagenen Tests sollte hier die Ursache schnell ersichtlich sein. Für die Überprüfung der COIN-Masken wurde entschieden, dass die Referenzdaten, der aktuelle Maskeninhalt und die Positionen der abweichenden Zeichen relevant sind. Standardmäßig zeigt der Report die Details immer zentriert formatiert in der Standardschriftart an. Um die Maskeninhalte optisch vergleichen zu können, ist diese Darstellung schlecht, da die ursprüngliche Darstellung der Masken verloren geht. Unter Verwendung des Scripting.Dictionary Objektes kann eine HTML-formatierte Darstellung erreicht werden. (vgl. Anhang)

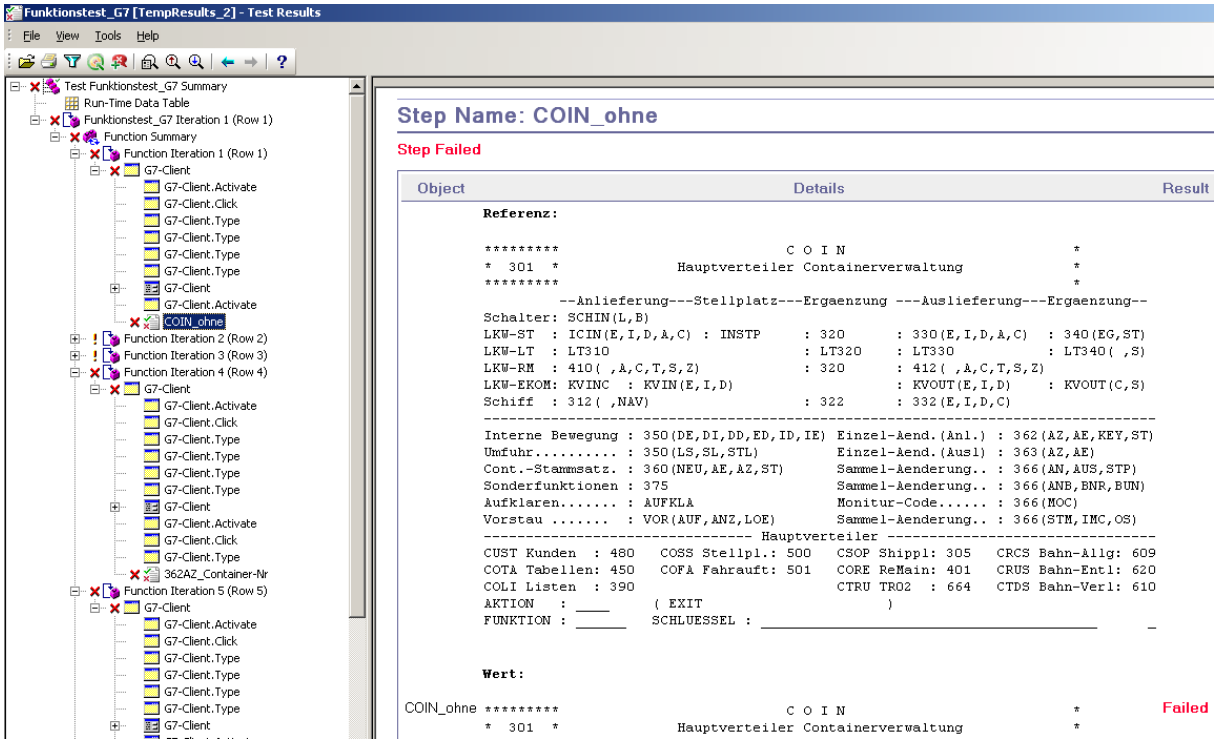


Abbildung 6-5: HP QC Reporter

Zusätzlich zu dem Eintrag im Reporter werden zwei Dateien erzeugt. Eine Datei enthält die aktuellen Referenzdaten, die andere den Maskeninhalt, der bei der aktuellen Ausführung angezeigt wurde. Als Namenskonventionen wurden

funktion_schlüssel_compTo_jjjmmtthhmmss.txt
funktion_schlüssel_err_jjjmmtthhmmss.txt

gewählt. Somit ist sicher dokumentiert, wann welcher Fehler aufgetreten ist. Es wäre durchaus vorstellbar, dass diese Dateien im HP Quality Center abgespeichert werden, um eine

konsistente Dokumentation zu ermöglichen und diese Dateien gegebenenfalls den entsprechenden Fehlern zuordnen zu können. Somit könnte das Defect Tracking unterstützt werden.

6.3.3 Analyse der Testergebnisse

Mit einer frühen Version des Neusystems wurde ein Testlauf mit 8 Aufrufen durchgeführt. Aufgrund des frühen Entwicklungsstadiums wurden einige Abweichungen aufgedeckt. Zuerst war aufgefallen, dass beim Testexport der Masken im Altsystem alle Eingabefelder Unterstriche enthalten, während im Neusystem an dieser Stelle Leerzeichen vorhanden sind. Der Komparator wurde daher so angepasst, dass der Vergleich eines Leerzeichens im Neusystem mit einem Unterstrich in den Referenzdaten keine Abweichung erzeugt.

Im Folgenden wird das Testergebnis einer Maske exemplarisch analysiert.

Referenzwert aus dem Altsystem für den Aufruf „481ANZ Reederei-Nr“:

```
*********                              C U S T                              *
*  481  * ANZEIGEN           K U N D E N S T A M M          EUROGATE   *
*********                    EVRE                                        *
Kunden-Nr: 01/01/00030119              Reederei-Nr : 025
Kurzname : KKL_____________________    Kurzort     : TOKYO___________
Branche  : __   _____________________  Leistu.-Ber.: __   _____________________
Kundenart: 0
                  Name 1 : "K"LINE_(EUROPE_LIMITED)D_____
                  Name 2 : C/O_"K"LINE___________________
   KUNDE          Name 3 : (DEUTSCHLAND)_GMBH____________
   AKTIV          Strasse: ANCKELMANNSPLATZ_1____________
                  Postf. : _______________
                  PLZ/Ort: 20537  HAMBURG_____________________
                  Land-Kz: D__     _______________

Telefon: _____________________   Kontakt-Person: _____________________
Telex  : _______________   Telefax: _______________   Teletex: _______________
Agentur - Reederei    : ____________________________________________________
Makler-Code           : ____
Inh./Geschaeftsfuehrer: _____________________________________________
Ansprechpartner       : _____________________________________________
AKTION   : ____     ( NEXT STOP EXIT           )
FUNKTION : ______   SCHLUESSEL: ____________________________________________
```

Ergebnis des Aufrufs „481ANZ Reederei-Nr“ im Neusystem:

```
*********                              C U S T                              *
*  481  * ANZEIGEN           K U N D E N S T A M M          EUROGATE    *
*********                    EVRE                                        *
Kunden-Nr: 01/01/00030119              Reederei-Nr : 025
Kurzname : KKL                         Kurzort     : TOKYO
Branche  : 00                          Leistu.-Ber.: 00
Kundenart: 0
                   Name 1 : "K"LINE (EUROPE LIMITED)D
                   Name 2 : C/O "K"LINE
   KUNDE           Name 3 : (DEUTSCHLAND) GMBH
   AKTIV           Strasse: ANCKELMANNSPLATZ 1
                   Postf. :
                   PLZ/Ort: 20537  HAMBURG
                   Land-Kz: D

Telefon:                          Kontakt-Person:
Telex  :                   Telefax:                    Teletex:
Agentur - Reederei    :
Makler-Code           :
Inh./Geschaeftsfuehrer:
Ansprechpartner       :
AKTION   :          ( NEXT STOP EXIT           )
FUNKTION :          SCHLUESSEL:
```

Es wurden automatisch folgende Abweichungen entdeckt:

```
Fehler in Zeile: 6 Position: 12
Fehler in Zeile: 6 Position: 13
Fehler in Zeile: 6 Position: 54
Fehler in Zeile: 6 Position: 55
```

Aus diesen Angaben und den fett formatierten Zeilen wird schnell ersichtlich, dass im Neusystem bei den Feldern „Branche“ und „Leistu.-Ber.“ jeweils „00“ angezeigt wird, während die Felder im Altsystem leer bleiben. Im Altsystem werden bei leeren Zahlfeldern automatisch die Nullen mit Leerzeichen ersetzt. Diese Einstellung wurde zu diesem Zeitpunkt im Neusystem noch nicht umgesetzt.

6.4 Wirtschaftlichkeit der Automatisierung

Jede Kombinationsmöglichkeit aus Funktion und verfügbaren Schlüssel ergibt einen Testfall. Insgesamt gibt es 400 Testfälle. Für den Test aller Funktionen werden 800 Aufrufe benötigt: 400 Aufrufe, um die Referenzdaten mit dem Altsystem zu erzeugen, und 400 Aufrufe, um die Ausgaben des Neusystems mit den Referenzdaten zu vergleichen. In einem Test mit 72 Aufrufen wurde ermittelt, dass für jeden Aufruf mit Erstellung von Referenzdaten bzw. der Vergleich mit den Referenzdaten 3 Sekunden benötigt werden. Somit benötigt ein automatisierter Test aller relevanten Funktion ca. 40 Min. Für die Testkosten wurde der der Stundensatz für einen Tester in Höhe von ca. 70 Euro angesetzt.

Testart	Art der Zeiterfassung	Anzahl Funktionsaufrufe	Testdauer	Dauer / Aufruf	Testkosten / Std.	Kosten / Test
manuell	gemessen	16	40 Min	2,5 Min.	70 €	175 €
automatisch	gemessen	144	420 Sek.	3 Sek.	70 €	8,16 €
manuell	berechnet	800	33 Std.	2,5 Min.	70 €	2.310 €
automatisch	berechnet	800	40 Min.	3 Sek.	70 €	46,66 €

Bei einem weiteren Testdurchlauf halbieren sich die Kosten für den automatischen Test, da die Referenzdaten nicht neu erstellt werden müssen, außer es wurden Änderungen an dem Datenbestand vorgenommen. Die manuelle Ausführung der Tests ist also bei der ersten Ausführung 50mal teurer als die automatisierte, danach jeweils 100mal teurer. Hinzu kommt, dass bei der Automatisierung eine automatische Dokumentation der Ergebnisse erfolgt und die Ergebnisse genauer sind, da keine Ermüdungserscheinungen den Test beeinflussen.

Aufgrund der kurzen Laufzeit von 40 Minuten für einen Test aller Funktionen kann dieser Test nach jedem neuen Versionstand der Software ausgeführt werden. Die Ergebnisse können so sehr schnell an die Entwickler weitergeben werden, wodurch der gesamte Entwicklungsprozess beschleunigt wird.

Analog zu Kapitel 3.4 Metriken lässt sich der ROI wie folgt berechnen.

	manueller Test	automatischer Test
Kosten für die Testfallerstellung	8.000 €	8.000 €
Toolkosten		3.000 €
Kosten Implementierung autom. Tests		6.000 €
Gesamtkosten Automatisierung		9.000 €
Kosten für einen Testzyklus	2.310 €	46,66 €
Anzahl der Zyklen pro Release	5	5
Testkosten pro Release	19.550 €	8.233,30 €
Einsparungen pro Release		11.316,70 €
ROI (Einsparung / Investition)		126 %

Die Kosten für die Testfallerstellung beinhalten alle Arbeiten zur Erstellung der Tabellen mit den Funktionsnamen und Schlüsseln sowie die Erarbeitung des Vorgehens für den Test. Die Implementierungskosten setzen sich aus den Kosten für das Erstellen der VBSkripte inklusive der Library für den Vergleich der Textdateien zusammen. Die hier erarbeiteten Funktionen können allerdings in anderen Tests weiter verwendet werden. Dies wurde in der Rechnung nicht berücksichtigt, genau wie der Umstand, dass sich die Kosten für weitere automatisierte Testausführungen halbieren, sofern die Referenzdaten nicht neu erstellt werden müssen.

6.5 Synthetische Daten

Synthetische Daten sind Testdaten, die speziell für die Tests angelegt werden und nicht aus dem Datenbestand, der in der Praxis entstanden ist, stammen. Das Erzeugen dieser Daten ist mit einem höheren Aufwand verbunden als das Kopieren von Daten aus einem Produktivsystem, bietet aber viele Vorteile.

6.5.1 Verwendung von synthetischen Daten

Bei Eurogate ITS ist es gängige Praxis, dass die Testumgebungen in regelmäßigen Abständen mit Daten aus dem Produktivsystem gefüllt werden. Es gibt verschiedene Argumente, die gegen eine solche Vorgehensweise sprechen. Bei automatisierten Tests kann es nötig sein, dass die Skripte an die aktuellen Daten angepasst werden müssen. Datensätze werden anhand einer Schlüsselnummer aufgerufen. Wenn der verwendete Schlüssel nun gerade keinem Datensatz zugeordnet ist, weil er in der Produktivumgebung nicht vergeben ist, schlägt der Test fehl. Wenn größere Testläufe geplant sind, die unbeobachtet über längere Zeit laufen sollen, kann ein erheblicher Aufwand entstehen, wenn alle Skripte an die neuen Daten angepasst werden müssen. Die Suche nach passenden Datensätzen bleibt auch beim manuellen Testen nicht aus, aber dem Tester wird es auffallen, wenn er einen bestimmten Datensatz nicht findet, und selbstständig einen anderen suchen. Dieses Verhalten muss in einem Testskript aufwendig implementiert werden.

Viele Tests benötigen einen bestimmten Zustand des Systems. Der Test kann nur dann sinnvoll ausgeführt werden, wenn ein definiertes Testszenario vorliegt. Arbeitet man mit Daten aus der Produktion, so kann es vorkommen, dass bestimmte Szenarien nicht vorhanden sind. Somit wird die erfolgreiche Ausführung dieser Tests zu einem zufälligen Ereignis. Die Änderung der Testdaten durch einen neuen Produktionsabzug erschwert die Nachvollziehbarkeit von Tests. Unter Umständen kann ein Fehler mit neuen Daten nicht reproduziert werden.

Diese Probleme können umgangen werden, indem synthetische Daten, also Daten die für den Test erstellt wurden und nicht in der Produktivumgebung entstanden sind, verwendet werden. Somit ist sichergestellt, dass unter einem bestimmten Index immer ein Datensatz mit bestimmten Eigenschaften vorhanden ist. Außerdem kann das System in einen definierten Zustand gebracht werden, um Tests in dem gewünschten Szenario ausführen zu können.

Es ist nicht notwendig, dass der gesamte Datenbestand aus synthetischen Daten besteht. Dies ist auch nicht immer möglich oder nur mit einem nicht vertretbaren Aufwand zu realisieren. Es ist ausreichend, alle relevanten Datensätze vor dem Testlauf zu erzeugen. Da diese Erzeugung immer gleich abläuft und verantwortlich für eine korrekte Testausführung ist, sollte sie automatisiert werden. Ein späterer Test mit Produktionsdaten sollte nicht vernachlässigt werden. Es besteht immer die Möglichkeit, dass Datenkonstellationen entstehen, die im Test nicht beachtet wurden.

6.5.2 Erzeugung von synthetischen Daten in COIN

Die zentrale Information in COIN sind Containerdaten. Es gibt bei Containerdaten sehr viele Variablen, die zu unterschiedlichen Geschäftsprozessen führen können. Bespiele für solche Angaben sind: Höhe, Länge, Leergewicht, Gesamtgewicht, Gefahrstoffe, Kühlcontainer, Standardcontainer, Zielhafen, Reeder usw.

Eine Möglichkeit der Verwendung von synthetischen Daten für COIN ist das Anlegen von Containern, die per Schiff exportiert werden sollen. Es erfolgt eine Vorabmeldung per EDI-Nachricht (COPRA Load), die beschreibt, welche Container auf das Schiff geladen werden sollen. Hier sind bereits die Staupositionen an Bord vermerkt. Die Container werden am Containerterminal per LKW, Zug oder per Schiff angeliefert. Diese Anlieferung wird in

COIN erfasst. Diese Daten bilden die Grundlage für verschiedene Aktivitäten in COIN und auch in dem operativen Planungssystem TOPX.

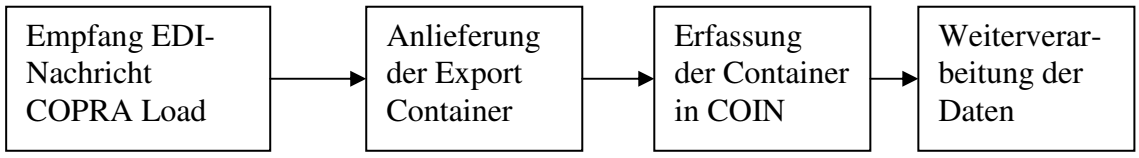

Empfang EDI-Nachricht COPRA Load
Anlieferung der Export Container
Erfassung der Container in COIN
Weiterverar-beitung der Daten

Abbildung 6-6: Geschäftsprozess Exportcontainer erfassen

Um auf diese Weise eine Datengrundlage für diverse Testausführungen zu erzeugen, wird eine EDI-Nachricht erzeugt, die mehr als 100 unterschiedliche Containerdaten enthält. Diese EDI-Nachricht wird über die entsprechende COIN-Schnittstelle importiert. Für die Erfassung der Anlieferung muss jeder einzelne Container mit einer Funktion in COIN erfasst werden. Diese Erfassung der Daten wurde mit QuickTest Professional automatisiert. Alle relevanten Daten der 100 Container wurden in einer Exceltabelle erfasst. Diese Daten werden dann mit einem Skript in die dafür vorgesehene Maske eingetragen. Da die EDI-Nachricht ebenfalls aus einer Excel-Tabelle erzeugt wird, ist die Erstellung der Datengrundlage für das Testskript denkbar einfach.

Für die automatisierte Eingabe der Containerdaten in die entsprechende COIN-Maske wird der Umstand genutzt, dass COIN automatisch mit dem Eingabecursor in das nächste Feld wechselt, wenn die maximale Anzahl an Zeichen in ein Eingabefeld eingegeben wurde. Der Cursor wird im ersten Feld positioniert, anschließend werden der Reihe nach alle Daten eingegeben. In der Data Table sind in der richtigen Reihenfolge alle Eingabefelder mit dem Wert, der eingegeben werden soll, und die Größe des Eingabefeldes angegeben.

	Wert	Size	C
1	Laenge	2	
2	Hoehe	2	
3	Typ	3	
4	Tara	4	
5	ZVA	1	
6	CSC	4	
7	MGW	5	
8	AnlDat	6	
9	Zeit	4	
10	ReedereiNr	3	
11	Brutto	5	
12	WA	1	
13	OS	1	

Abbildung 6-7: HP QTP Data Table Container anlegen

Somit besteht der Quellcode für die Eingabe der Daten nur noch aus wenigen Zeilen. Die diskreten Werte stehen wieder in dem Tabellenblatt „global“ der Data Table und werden von dort übernommen. Bei der Eingabe werden die Werte mit Leerzeichen aufgefüllt, damit die maximale Eingabegröße erreicht wird und der Cursor in das nächste Feld springt. Dies hätte auch mit dem Senden eines Tabulators realisiert werden können, so wird aber sichergestellt, dass alte Werte, die schon im Feld stehen, überschrieben werden. Wenn kein Wert für die

Eingabe in der Data Table hinterlegt ist, wird das Feld mit „TAB“ übersprungen. Hier werden keine Leerzeichen eingegeben, damit keine Standardwerte überschrieben werden.

```
Dim Wert
wert =  DataTable( DataTable("Wert",dtLocalSheet),dtglobal)
If wert = "" Then
        Window("ExceedWindow").Type  mictab
else
Window("ExceedWindow").Type  wert &Space(datatable("Size",dtLocalSheet)-len(wert))
end if
```

Mit diesem Code kann jede Eingabemaske gesteuert werden. Die Ablauflogik ist in der Data Table festgelegt. Dort müssen die Eingabefelder in der richtigen Reihenfolge angegeben werden. Dieser Code wird für jede Zeile in der Data Table wiederholt. Die Zeilen des globalen Datenblattes der Data Table agieren bei diesem Ablauf als äußere Schleife. Jede Zeile stellt einen Containerdatensatz dar, der über die COIN-Maske angelegt werden soll.

Schluessel	Laenge	Hoehe	Typ	Tara
robe0219100	20	86	dc	2000
robe0219101	40	86	dc	2000
robe0219102	20	96	rf	2000
robe0219103	40	96	rf	2000

Abbildung 6-8: HP QTP Data Table globales Datenblatt

6.6 Testen von Geschäftsprozessen

Das Aufrufen aller vorhandenen COIN Funktionen und Masken ist nicht ausreichend, um die Korrektheit der COIN-Migration sicherzustellen. So wird beispielsweise nicht überprüft, ob eingegebene oder veränderte Daten tatsächlich in der Datenbank gespeichert oder aktualisiert werden. Hierzu sind Tests nötig, die aus mehr als nur dem Aufrufen einer Funktion bestehen. Es werden mindestens drei Schritte benötigt.

1. Maske aufrufen
2. Daten eingeben
3. Sicherstellen, dass diese Daten gespeichert wurden

Dies wäre ein sehr einfacher Geschäftsprozess, da nur eine Programmmaske beteiligt ist. In QuickTest Professional wurde das Anlegen von Containerdatensätzen realisiert. Dieser Prozess wird durch Eingaben in die entsprechende Maske umgesetzt. Um zu überprüfen, ob die Container tatsächlich gespeichert wurden, werden nach dem Anlegen die entsprechenden Datensätze angezeigt. Alternativ könnte an dieser Stelle eine Datenbankabfrage per SQL erfolgen. Für die COIN-Migration kann zur Validierung der Ergebnisse wieder der Vergleich über die Referenzdaten erfolgen. Das Anlegen des gleichen Containerdatensatzes in beiden Systemen muss auch zu dem gleichen Ergebnis führen.

Nach diesem Prinzip lassen sich auch komplexere Vorgänge automatisieren. Es müssen die jeweiligen Eingabemasken aufgerufen und mit Daten gefüllt werden. Anschließend können die Ergebnisse über geeignete Anzeigemasken geprüft werden. Durch die Verwendung von

synthetischen Daten lassen sich die Überprüfungen einfach implementieren, da genau vorhersagbar ist, welches Ergebnis erzeugt werden muss.

6.7 Aktueller Stand und weiteres Vorgehen

Mit dem automatisierten Aufruf aller relevanten Funktionen und Masken können viele Fehler, die in der Entwicklungsphase des Migrationsprojektes auftreten, schnell gefunden werden. Aufgrund der kurzen Dauer der Ausführung können diese Tests in regelmäßigen Abständen durchgeführt werden. Somit ist ein kurzfristiges Reporting an die Entwickler möglich, die ihrerseits schnell auf Probleme reagieren können. Es wurde gezeigt, dass die Automatisierung in diesem Bereich sowohl einen finanziellen als auch einen qualitativen Vorteil bringt. Leider kann nicht weiter auf aufgedeckte Fehler eingegangen werden, da sich die Auslieferung des Neusystems aufgrund verschiedener Probleme verzögert hat.

Es wurde dargestellt, dass mit relativ wenig Aufwand synthetische Daten für weitere Tests erzeugt werden können und dass mit dem Know-how über Geschäftsprozesse diese auch als Testfälle in QuickTest Professional implementiert werden können. Das entstandene Automatisierungskonzept ist entsprechend erweiterbar, und es können komplexere Tests realisiert werden.

Nach einer erfolgreichen Migration wird das Neusystem von den Entwicklern bei Eurogate weiterentwickelt und gegebenenfalls an neue Geschäftsprozesse angepasst. Mit dem vorhandenen Automatisierungskonzept können diese Modifikationen mit einem automatisierten Regressionstests validiert werden. Es würden dann nicht mehr die Ausgabe des Altsystems mit der des Neusystems verglichen werden, wie es bei der Migration der Fall ist, sondern es würden die unterschiedlichen Versionsstände gegeneinander geprüft werden. An dieser Stelle sollte evaluiert werden, ob es einen effektiven Nutzen gibt, wenn die Überprüfung von eingegebenen Daten und das Anlegen von synthetischen Daten mit SQL-Statements an die Oracle-Datenbank erfolgt. QuickTest Professional benötigt hierzu eine ODBC-Verbindung zu der Datenbank. Da diese Verbindung mit dem Unisys-Host nicht möglich ist, wurde auf SQL-Statements für die Tests während der Migration verzichtet.

7 Automatisierung der TOPX Tests

In diesem Kapitel wird analysiert, wie die Ausführung der Tests für das System TOPX mit Automatisierungen unterstützt werden kann. Es wird zunächst der Ist-Zustand ermittelt und dem Zielzustand gegenüber gestellt. Um den optimalen Zielzustand zu erreichen, sind bestimmte Bedingungen Voraussetzung. Sind alle diese Bedingungen erfüllt, kann eine lehrbuchartige Umsetzung erfolgen. Dies betrifft zum Beispiel die Testfallermittlung. Die Testfälle für Black Box Tests sollen aus der Spezifikation abgeleitet werden. Dies setzt eine ausreichend detaillierte Spezifikation voraus. Ist diese nicht vorhanden, wird es selten möglich sein, schnell eine entsprechende Spezifikation zu erstellen. Also müssen die Testfälle mit anderen Methoden ermittelt werden. Bei der weiteren Analyse werden derartige Probleme der Praxis dargestellt. Es soll ein Konzept gefunden werden, mit dem trotz dieser Probleme schrittweise ein funktionierender Testprozess entstehen kann.

7.1 Ist-Zustand

Es werden Tests für aktuell angekündigte Releases von TOPX durchgeführt. Diese Releases basieren auf Change Requests (Änderungsanforderungen), die in der Zeit nach der Einführung der Software am Container Terminal Hamburg erfasst wurden. Viele dieser Änderungen sind Korrekturen an den umgesetzten Geschäftsprozessen, da diese nicht den gewünschten Vorgehensweisen entsprechen. Hinzu kommen noch diverse Bugfixes. Die Durchführung der Tests übernimmt noch der externe Dienstleister unter Anleitung durch das Testmanagement von Eurogate ITS.

7.1.1 Testfallermittlung

Die Ermittlung der Testfälle wird von Mitarbeitern aus den Fachabteilungen vorgenommen. Es wird zu den Änderungsanforderungen ermittelt, welche Bereiche von TOPX betroffen sind und dann entschieden, was getestet werden muss. Die Analyse erfolgt anhand von drei Kriterien: Welchen Einfluss hat die Änderungen auf Geschäftsprozesse, benachbarte Systemkomponenten und Schnittstellen zu Umgebungssystemen? Die Fachabteilungen, die mit den betroffenen Systemteilen arbeiten, beschreiben die Testfälle und Testszenarien. Ein Testszenario wäre zum Beispiel das Löschen (Entladen) eines Schiffes mit einer bestimmten Ladung. Dazu kann es mehrere Testfälle geben, die zum Beispiel die unterschiedliche Behandlung von verschiedenen Containertypen überprüfen. Kühlcontainer bekommen einen Stellplatz mit einem Stromanschluss, und bestimmte Gefahrstoffcontainer dürfen nicht über- oder nebeneinander gelagert werden.

Die Geschäftsprozesse, die diesen Tests zugrunde liegen, sind in einer Form dokumentiert, die ein Mitarbeiter der entsprechenden Fachabteilung nachvollziehen kann. Ein Mitarbeiter, dem diese Abläufe völlig fremd sind, ist nicht in der Lage, anhand dieser Dokumentation einen Testfall zu erstellen. Dafür fehlt der tiefgehende Detailgrad dieser Dokumentation. Zu diesen Tests, die aktuelle Änderungen überprüfen, werden Regressionstests ausgeführt, die sicherstellen sollen, dass durch die Änderungen keine bestehenden Funktionalitäten

beeinträchtigt werden und keine, bisher eventuell maskierten Fehler das Hauptgeschäft von Eurogate beeinträchtigen.

7.1.2 Testdokumentation

Die Testfälle, die Testausführungen und die Änderungsanforderungen werden im HP Quality Center dokumentiert. Ermittelte Fehler werden in „Intrexx“ erfasst. Intrexx wird eigentlich als Ticket System für den User Help Desk verwendet. Hier können aus erfassten Tickets gegebenenfalls Defects erzeugt werden. Der Hersteller von TOPX hat Zugriff auf dieses System und kann die bestehenden Defects einsehen. Dieses System wurde bereits vor der Einführung des HP Quality Centers verwendet. Es sind also Fehler in diesem System hinterlegt, die vor der Einführung des HP Quality Centers aufgetreten sind, aber noch bearbeitet werden. Neue Fehler werden weiterhin in Intrexx erfasst, aber über eindeutige Indizes im Quality Center referenziert. So muss der Lieferant nicht mit zwei Systemen arbeiten, aber trotzdem wird die Funktionalität des HP Quality Centers in Bezug auf Defect Tracking und Analysemöglichkeiten des Testprozesses voll ausgenutzt. Für zukünftige Projekte ist geplant, dass nur noch das HP Quality Center verwendet wird.

7.1.3 Testpriorisierung

Zu jedem Release wird betrachtet, in welchem Funktionsbereich die meisten Änderungen vorgenommen wurden. Diese Bereiche werden generell am stärksten mit Testfällen überdeckt. Die Fachabteilungen, die sich in den jeweiligen Bereichen auskennen, priorisieren die Testfälle innerhalb dieser Bereiche. Die Kritikalität wird in 5 Kategorien unterteilt, von 1 (kritisch) bis 5 (gering). Die Kategorie wird danach bemessen, wie schnell und wie stark ein Ausfall des entsprechenden Bereiches Auswirkungen auf die Produktion hat.

7.1.4 Testausführung

Die Tests werden von Testern des externen Dienstleisters, Mitarbeitern aus den Fachbereichen bei Eurogate und Mitarbeiten von Eurogate ITS durchgeführt. Diese Mitarbeiter werden für die Testausführung von ihren eigentlichen Aufgaben freigestellt. Die Ausführung der Tests für ein neues Release kann bis zu 3 Wochen oder länger dauern. Die Mitarbeiter führen diese Tests in dieser Zeit entweder an 2 oder 3 Tagen pro Woche oder in Vollzeit aus. Die Tests werden anhand der im HP Quality Center dokumentierten Testschritte manuell ausgeführt. Die Ergebnisse werden ebenfalls im HP Quality Center hinterlegt. Es gibt keine Unterstützung durch automatisierte Prozesse. Als Testdaten werden für gewöhnlich Produktionsdaten genommen. Dies führt häufig dazu, dass zu dem jeweiligen Testszenario passende Datensätze zuerst im System gesucht werden müssen. Eventuell müssen diese Daten zuerst von Hand angelegt werden. Besonders wenn sehr viele oder sehr spezielle Daten benötigt werden, führt dies zu erheblichen Verzögerungen.

7.2 Zielzustand

Das Hauptziel ist, dass alle Tests ohne den externen Dienstleister ausgeführt werden. Sämtliche Testaktivitäten sollen von Eurogate ITS erledigt werden. Für den Test bedeutet

dies, dass er mit weniger Personal ausgeführt werden soll, ohne dass dies eine Verlängerung der Testausführung oder eine Verringerung der Qualität zur Folge hat. Um das zu erreichen, müssen die manuellen Tests so gut wie möglich durch eine Automatisierung ersetzt oder unterstützt werden.

Als Vison kann man sicherlich einen vollständig automatisierten Regressionstest nennen, der die Funktionsfähigkeit sämtlicher Geschäftsprozesse sicherstellt. Dazu ist nicht nur eine Automatisierung der Abläufe in TOPX notwendig, sondern auch in COIN. Es müssen also Tests ausgeführt werden, die mindestens zwei Programme bedienen und auch noch mit zwei verschiedenen Automatisierungstools ausgeführt werden müssen. Durch entsprechende Funktionalitäten im HP Quality Center ist diese Anforderung zumindest technisch durchführbar. Leider ist hierfür ein nicht unerheblicher Implementierungsaufwand nötig. Deshalb ist es sinnvoll, zuerst zu analysieren, welche Aufgaben mit weniger Aufwand automatisiert werden können und trotzdem entscheidende Verbesserungen bewirken.

7.3 Skizzierung einer möglichen Umsetzung

Eine mögliche Aufgabe, die mit automatischen Abläufen erledigt werden kann, ist die Testvorbereitung. Hierbei handelt es sich um das Erzeugen von Testdaten und Testszenarien, die Durchführung von Tests und die automatische Überprüfung von Ausgaben der getesteten Software mit den erwarteten Ausgaben. (vgl. Kap.3 Testautomatisierung)

Für die Einführung von automatischen Tests ist es sinnvoll, mit einem kleinen Projekt zu beginnen, um Erfahrungen zu sammeln. Leider gibt es bei Eurogate ITS kein kleines Projekt, das als Pilotprojekt für die Automatisierung benutzt werden kann. Die Aufgaben, die für das COIN-Projekt realisiert wurden, können zwar als Pilot angesehen werden, aber TOPX hat andere technische Eigenschaften, die ein anderes Vorgehen und ein anderes Tool erfordern.

Um trotzdem einen Einstieg zu finden, sollten Aufgaben automatisiert werden, deren Implementierung nicht zu aufwendig erscheint. Das Ergebnis sollte aber einen Gewinn für den Testprozess darstellen. Die Automatisierung ganzer Tests scheint hier zu aufwendig. TOPX verwendet eine GUI, die aus mehreren Fenstern besteht, es werden Menüleisten eingeblendet, deren Funktionalität abhängig von aktiven Fenster ist, es gibt viele Listen und etliche Informationen, die nur grafisch dargestellt werden. Zum Beispiel die Belegung der Bays (Laderäume) wie in Abbildung 7-1. Um mit TOPX zu arbeiten, werden zwei Monitore verwendet, damit trotz der Vielzahl an Fenstern, Masken und Tabellen die Übersicht erhalten bleibt. Diese Gesamtkomplexität macht eine GUI-Test-Automatisierung zu einer echten Herausforderung, und zwar besonders, wenn eine zuverlässige Überprüfung des Verhaltens von TOPX erfolgen soll.

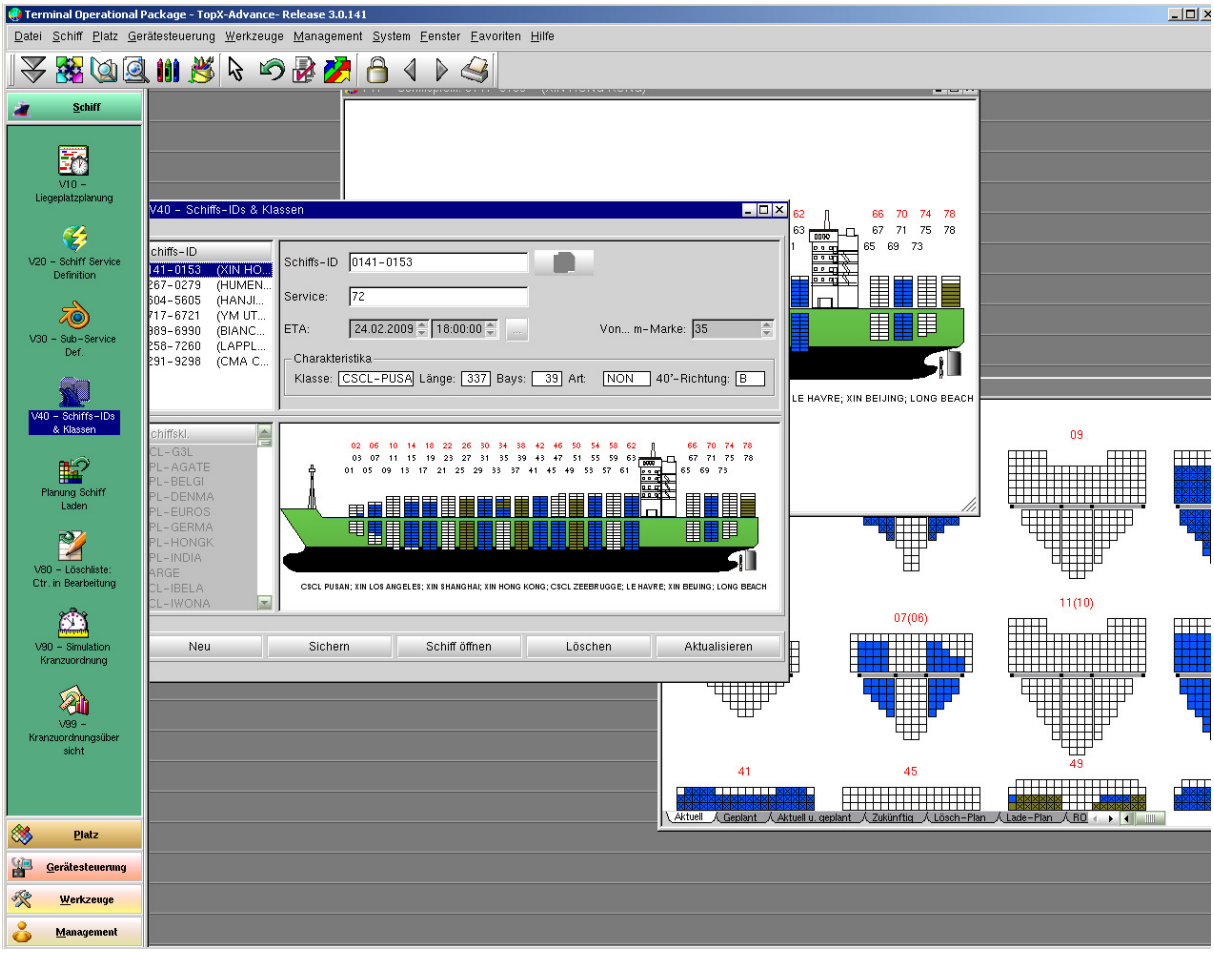


Abbildung 7-1: TOPX mit geöffneter Schiffsansicht

Um einen Testfall durchzuführen, wird momentan sehr viel Zeit darauf verwendet, passende Daten für den Testfall zu finden. Soll zum Beispiel ein Testfall mit einem Gefahrgutcontainer, der von einem Schiff gelöscht wird, ausgeführt werden, muss erst in dem Datenbestand ein entsprechender Container gesucht werden. In manchen Fällen kann dies länger dauernals die Ausführung des eigentlichen Tests. Während des Testens kann es auch vorkommen, dass die Anzahl der Container, die im System vorhanden sind, nicht ausreichen, um alle geplanten Tests auszuführen. Hier kann die Verwendung von synthetischen Daten eine große Hilfe sein. (vgl. Kap. 6.5 Synthetischen Daten)

Die Container, die in TOPX verwendet werden, werden über das COIN-System erfasst und über Schnittstellen in die Datenbank von TOPX importiert. Allerdings sind teilweise weitere Schritte nötig, um die Daten so zu manipulieren, dass sie für die TOPX-Tests verwendet werden können. Solche Schritte können zum Beispiel die Zuordnungen zu bestimmten Geräten (Containerbrücken, Schiffen o.ä.) sein. Ein Ansatz wäre, die Daten direkt in die Datenbank von TOPX zu schreiben. Dies wäre per SQL-Statement möglich. Allerdings werden durch bestimmte GUI-Aktionen in TOPX diverse Schalter in der Programmlogik gesetzt, so dass reine Änderungen an der Datenbank möglicherweise nicht die gewünschten Ergebnisse liefern und gegebenenfalls zu Inkonsistenzen in der Datenbank führen würden. Da der Hersteller von TOPX auch das Datenbankschema entwickelt hat, ist das Know-How bei Eurogate nicht ausreichend, um zuverlässige SQL-Statements zu erzeugen.

Ein Lösungsansatz wäre also die Erzeugung der Daten wie in Kap. 6.5.2 Erzeugung von synthetischen Daten in COIN umzusetzen und weitere Schritte in TOPX mit einer GUI-Automatisierung ablaufen zu lassen. Diese Aufgabe sollte mit dem Tool Squish von Froglogic erfolgen. (vgl. Kap. 5.6 Froglogic Squish)

Der so erreichte Vorteil wäre, dass in den Testfallbeschreibungen dokumentiert werden würde, welche Container für den entsprechenden Test verwendet werden müssen. Dies spart zum einen die Zeit für das Suchen der Datensätze ein; zum anderen wird auch sichergestellt, dass der Test tatsächlich mit den Daten ausgeführt wird, die alle Kriterien erfüllen, die für eine korrekte Testausführung nötig sind. Die Erzeugung der synthetischen Daten könnte auch manuell ausgeführt werden, dabei besteht aber die Gefahr, dass bei der Eingabe Fehler gemacht werden. Ein Ursache können auch hier Ermüdungserscheinungen sein, wenn eine große Anzahl an Datensätzen manuell eingegeben werden muss. Da diese Daten für jeden Testdurchlauf benötigt werden, handelt es sich um eine wiederkehrende Tätigkeit. Es sind alle Kriterien erfüllt, die eine Automatisierung rechtfertigen.

Nicht alle Daten, die für die Tests benötigt werden, sind Containerdaten. Diese werden aber mit Abstand am häufigsten benötigt. Deshalb würde das Erstellen von synthetischen Containerdaten schnell zu positiven Effekten führen. Das reine Erzeugen der Daten kann anschließend dazu ausgebaut werden, im Test häufig verwendete Testszenarien zu erzeugen. Dies können Szenarien sein, die in der Praxis oft vorkommen oder die eher ungewöhnlich sind und häufig zu Fehlern führen. Besonders wichtig wären die Szenarien, mit denen die Funktionalitäten geprüft werden, die für das Hauptgeschäft von Eurogate wichtig sind. Diese Szenarien lassen sich aus den Testdokumentationen ableiten.

Als nächstes sollte die schrittweise Automatisierung der Regressionstests, die das Hauptgeschäft von Eurogate absichern, folgen. Hier sollte eine genaue Analyse erfolgen, welche Tests sich mit möglichst wenig Aufwand automatisieren lassen. Mit diesen Tests sollte begonnen werden, um Erfahrungen zu sammeln, wie erfolgreich die Automatisierung ist. Mit diesen ersten Tests muss eine generelle Vorgehensweise für die Automatisierung festgelegt werden. Nach Möglichkeit sollten dabei Standards entwickelt werden. Diese Standards können recht umfassend sein. So sollte festgelegt werden, wie die Testfälle und Testergebnisse dokumentiert werden, aber auch die Art und Weise, wie Tests automatisiert werden. Hier kann unter anderem ein Coding-Style für die Skripte festgelegt werden oder bestimmte Arten der Überprüfung. Somit wird sichergestellt, dass gewonnene Erfahrungen auch einen positiven Effekt auf den weiteren Testprozess haben.

Aus nützlichen Funktionen, die dabei entwickelt werden, kann ein Testframework erzeugt werden. Die Geschäftsprozesse, die diesen Tests zu Grunde liegen, müssen in einer Form dokumentiert werden, die auch von Mitarbeiten verstanden wird, die nicht in den entsprechenden Fachbereichen tätig sind. So ein Mitarbeiter wäre unter anderem der Testautomatisierer. Da die Dokumentation in dieser Tiefe noch nicht existiert, sollten die Prozesse ausführlich dokumentiert werden, die zuerst automatisiert werden sollen. Eine Zusammenarbeit zwischen der Fachabteilung und dem Testautomatisierer ist unerlässlich.

Eine Erfassung verschiedener Metriken wäre ebenfalls ratsam. Durch den konsequenten Einsatz des Quality Centers von HP können einige wichtige Punkte, wie die erreichte Testabdeckung, auf Knopfdruck ermittelt werden. Es könnte somit eine Qualitätssicherung für den Testprozess entstehen. Eine ausführliche Gegenüberstellung der Metriken vor und nach

der Übernahme des Tests durch Eurogate ITS kann ein wichtiger Punkt im Reporting gegenüber der Geschäftleitung sein.

Auch mit der hier dargestellten Vorgehensweise wird die Umsetzung der Vision nicht in kurzer Zeit möglich sein – es kann sogar Jahre dauern. Die Analyse und die Verbesserungen am Testprozess sind ein Vorgang, der niemals enden wird. Es wird neue Testfälle und neue Programme geben, die getestet werden sollen. Mit einem gut strukturierten und geplanten Testprozess können diese Anpassungen minimiert werden. Ein gut funktionierender automatisierter Testprozess ist keine Selbstverständlichkeit. 40 bis 50 Prozent der Unternehmen, die einen solchen Prozess realisieren wollten und bereits ein Tool zur Automatisierung erworben haben, nutzen es nicht, da die Umsetzung nicht konsequent erfolgt ist.
[vgl. Fewster1999 S. 299-300]

8 Zusammenfassung und Ausblick

In den Kapiteln 2 Testgrundlagen und 3 Testautomatisierung wurde dargestellt, was beim Testen von Software und bei der Automatisierung wichtig ist. Diese Erkenntnisse wurden in den späteren Kapiteln umgesetzt und auf die beiden hier vorgestellten Projekte angewandt. Die Ergebnisse und Hinweise zum weiteren Vorgehen können den entsprechenden Kapiteln 6.7 Aktueller Stand und weiteres Vorgehen und 7.3 Mögliche Vorgehensweise für die Umsetzung entnommen werden.

Während der Entstehung dieser Arbeit wurde deutlich, wie wichtig eine gute Dokumentation der Anforderungen ist. Sicherlich ist dies ein Punkt, der in vielen Fachbüchern angemerkt wird, aber leider in der Praxis nicht konsequent umgesetzt wird. In der Entstehung bedeutet Dokumentation meistens eine Mehrarbeit, die zudem noch sehr unbeliebt ist. Aber eine gute Dokumentation nachträglich zu erstellen, ist erheblich aufwendiger, und meistens ist gerade dann, wenn bemerkt wird, dass die Dokumentation nicht ausreichend ist, Zeit ein knappes Gut.

Es wurde ebenfalls deutlich, dass mit der Testautomatisierung erhebliche Verbesserungen am Testprozess erreicht werden können. Es können effektivere Tests ausgeführt werden, es können mehr Tests ausgeführt werden, oder die Testphase kann verkürzt werden, gegebenenfalls auch mit dem Einsatz von weniger Ressourcen. Aber der anfallende Aufwand für die Automatisierung darf nicht unterschätzt werden. Es kostet Zeit, die richtigen Tests für die Automatisierung zu identifizieren und die richtigen Tools für die Testausführung auszuwählen. Dies sind zwei elementare Aspekte, die zunächst keinen produktiven Nutzen erzeugen, aber das Grundgerüst für die weitere Arbeit bilden. Werden an dieser Stelle Fehler gemacht, kann dies bereits immense Auswirkungen auf den Erfolg beziehungsweise den Misserfolg des Automatisierungsprozesses haben. Dem Testautomatisierer sollte daher ausreichende Zeit zur Verfügung stehen, sich einen gründlichen Überblick über die vorhandenen Softwareprojekte zu verschaffen. An dieser Stelle wird wieder die Bedeutung einer zuverlässigen Dokumentation sichtbar. Es wird sicherlich trotzdem notwendig sein, dass sich der Testautomatisierer bei den entsprechenden Kollegen ausgiebig über die Projekte informiert, aber mit einem fundierten Basiswissen können diese Gespräche erheblich effektiver ausfallen. Sollten an dieser Stelle Lücken in der Dokumentation auffallen, wäre es wohl auch ein guter Zeitpunkt, diese zu schließen. Dabei sollte das Erstellen oder Überarbeiten der Dokumentation nicht die Aufgabe des Testautomatisierers sein.

Ebenfalls wichtig für den Erfolg der Testautomatisierung ist die allgemeine Akzeptanz. Wenn die Mitarbeiter die Besprechungen mit dem Testautomatisierer eher als Zeitverschwendung für eine „neue Spielerei“ ansehen, werden sie vermutlich nicht mit der notwendigen Sorgfalt vorgehen und versuchen, die Gespräche kurz zu halten. Dabei kann es unter Umständen vorkommen, dass wichtige Informationen nicht weitergegeben werden. Ein gutes Mittel, um diese Akzeptanz zu erzeugen, ist, die Fortschritte regelmäßig zu präsentieren. Viele Kollegen haben sich vielleicht noch nie mit dem Thema beschäftigt. Wenn das erste Mal gesehen wird, wie eine Software von einem Automatisierungstool bedient wird, kann das eine faszinierende Wirkung haben, die den Glauben stärkt, dass diese „kleine Spielerei“ erfolgreich sein kann. Natürlich sollte dabei darauf geachtet werden, dass keine falschen oder überzogenen Erwartungen erzeugt werden, damit kurze Zeit späte keine große Enttäuschung folgt.

Die Implementierung von Automatisierungsskripten ist Softwareentwicklung. Es werden Sprachen verwendet wie Visual Basic Script oder JavaScript, die alle Möglichkeiten einer Programmiersprache bieten. Das bedeutet wiederum, dass sie auch alle Fehlermöglichkeiten einer Programmiersprache enthalten. Alle Skripte müssen sorgfältig programmiert, dokumentiert und getestet werden. Um den Aufwand hier zu minimieren, ist es ratsam, einen „keep-it-simple“ Ansatz zu verfolgen. Wenn ein erhöhter Programmieraufwand benötigt wird, um einen Fehler zu entdecken, der unbedeutend ist, dann sollte man an dieser Stelle vielleicht von der Prüfung dieses Fehlers absehen. Dies ist ein Mittel, um die Wartbarkeit der Testskripte zu erhalten. Häufig benötigte Funktionalitäten sollten in Libraries ausgelagert werden. Diese können in weiteren Tests verwendet werden, und es kann eine bessere Strukturierung der Skripte erfolgen. So kann nach und nach ein Testframework entstehen, auf dessen Basis sich später schneller zuverlässige Testskripte erzeugen lassen.

Aus Sicht des Unternehmens kommt sicherlich die Frage auf, welche Anforderungen an einen Testautomatisierer zu stellen sind. Aus den vorhergehenden Absätzen kann abgeleitet werden, dass der Testautomatisierer über gute Kenntnisse in der Softwareentwicklung verfügen muss. Er muss sorgfältig programmieren können, um zuverlässige Testskripte zu erstellen, und er muss sich mit verschiedenen Systemarchitekturen auskennen, um die zu testende Software zu verstehen. Nur mit einem guten Überblick über die allgemeine Funktionsweise von Software können Skripte erstellt werden, die die wesentlichen Aspekte überprüfen und trotzdem robust ablaufen. Dies impliziert die Fähigkeit, sich in komplexe Projekte einzuarbeiten und sich einen fundierten Überblick zu verschaffen. Darüber hinaus sollte der Testautomatisierer in der Lage sein, die gängigen Testmethoden auf die Projekte anzuwenden.

Außerdem werden sehr gute kommunikative Fähigkeiten benötigt. Die Ergebnisse der Automatisierung müssen gut präsentiert werden, um Vertrauen in die Automatisierung zu schaffen. Es müssen viele Details mit den Kollegen geklärt werden. Diese Gespräche müssen ausführlich sein, dürfen aber nicht zu einem Störfaktor werden. Von einem kompetenten Gesprächpartner fühlen sich die Kollegen sicherlich weniger schnell gestört und sind eher bereit, eine ausführliche Unterstützung zu bieten.

Der Testautomatisierer muss selbst überzeugt sein, dass die Testautomatisierung eine wichtige Sache ist. Nur so kann er auch andere davon überzeugen. Es sollte also niemand aus dem Unternehmen die Automatisierung übernehmen müssen, nur weil er seine eigentlichen Aufgaben ihm dafür etwas Zeit lassen. Ab einer gewissen Größe und Komplexität der Projekte sollte mindestens eine Person in Vollzeit für die Automatisierung verantwortlich sein. Es empfiehlt sich eine zweite Person, zumindest in Teilzeit, mit der Automatisierung zu beauftragen. Somit können zwischen den Personen Absprachen über weiteres Vorgehen, Techniken etc. getroffen werden. Ebenfalls ließe sich so eine Urlaubsregelung treffen.

Testautomatisierung ist, je nach Unternehmen, eine nicht unwesentliche Investition, die einen großen Nutzen bieten kann, sofern sie sorgfältig betrieben wird. Wurde die Entscheidung für eine Automatisierung getroffen, muss sie konsequent weiter verfolgt werden. Automatisierungsskripte, die ein Mitarbeiter nebenbei erstellt hat, während er mit anderen Aufgaben und Projekten beschäftigt ist, fehlt es an Zuverlässigkeit. Wenn man sich nicht auf die Richtigkeit der automatisierten Tests verlassen kann, ist es vielleicht sogar notwendig, sich in einem manuellen Test davon zu überzeugen oder die Testläufe mit einer aufwendigen Qualitätssicherung zu überprüfen. Dann wiederum hätten die Kosten für die Automatisierung gleich gespart werden können. Es besteht die Gefahr, dass das Bestreben, eine

Automatisierung umzusetzen, nachlässt und niemand mehr die dafür gekauften Tools verwendet.

Die Komplexität der Projekte bei Eurogate ist so hoch, dass es ratsam ist, ein Team ausschließlich für die Testautomatisierung zu beschäftigen. Für aufwendige Regressionstests werden Tests benötigt, die mit zwei stark unterschiedlichen Programmen arbeiten, den hier vorgestellten Anwendungen COIN und TOPX. Für eine weitgehende Automatisierung auf Basis des bisher Erreichten ist es notwendig, ein Testframework zu schaffen, das die Automatisierungstools QuickTest Professional und Squish mit Hilfe des Quality Centers miteinander verbindet. So kann eine Basis an Funktionen und Erfahrungen entstehen, die auf andere Projekte und Standorte übertragen werden können. Da an den Container Terminals von Eurogate verschiedene Softwaresysteme eingesetzt werden, wird es nicht möglich sein, die in Hamburg erstellten Tests ohne Anpassungen zu übernehmen. Es müssen größere Anpassungen vorgenommen werden oder sogar große Teile der Skripte neu implementiert werden. Die Realisierung von automatisierten Tests für mehrere Standorte ist also als langfristiges Projekt anzusehen, wobei neue Softwareversionen häufig auch Anpassungen an den automatisierten Tests bedeuten werden.

In der Endphase der COIN Migrationstests, wie sie hier beschrieben sind, konnten recht schnelle und deutliche Erfolge produziert wurden, die die allgemeine Akzeptanz von automatisierten Tests gesteigert haben. Hier war eine Verbesserung im Vergleich zur Anfangsphase deutlich bemerkbar. Nach ersten Lieferungen des Neusystems wurde aber auch deutlich, dass für einen umfassenden automatischen Test noch viel Arbeit investiert werden muss. Diese Arbeit wird zum Teil durch Details verursacht, in denen sich die Neu- und Altsysteme gewollt unterscheiden, was aber erst im Lauf des Projektes ersichtlich wurde. Auch wurde deutlich, dass weitere Verbesserungen in der Dokumentation der Anwendungen COIN und TOPX benötigt werden. Aber trotz dieser Schwierigkeiten konnten schnell Fehler in dem Neusystem identifiziert werden – sogar mehr als ursprünglich erwartet. Diesen ersten Erfolge und positiven Erfahrungen sollten zeitnah ausgenutzt werden, um die Testautomatisierung weiter voran zu treiben.

Im Projektplan der COIN-Migration ist vorgesehen, dass der gesamte Test von Eurogate ITS ausgeführt wird. Der knappe Zeitplan kann nur eingehalten werden, indem eine effektive Testautomatisierung eingesetzt wird. Deswegen sollte zuerst die Umsetzung dieser Tests im Mittelpunkt stehen. Eine parallele Implementierung von automatisierten Tests für TOPX wird mit den aktuellen Ressourcen nicht möglich sein.

Für die COIN-Migration bedeutet dies, dass nun Testfälle erstellt werden sollten, die Geschäftsprozesse umfassen; beginnend mit einfachen Tests, wie dem Erstellen eines Containerdatensatzes, Anzeigen der Daten, Ändern der Daten und Löschen der Daten. Somit können alle Arten des Datenbankzugriffs auf diesen Bereich sichergestellt werden. Dieser Ablauf sollte automatisiert werden. Er kann anschließend so modifiziert werden, dass nicht nur ein Container bearbeitet wird, sondern sehr viele. Damit ließe sich eine bessere Aussage über die Zuverlässigkeit des Systems treffen. Nach und nach werden auf diese Weise weitere Testfälle hinzukommen, die schließlich zu einem ausführlichen Regressionstest für COIN werden. Da diese Testfälle auf das Neusystem angewendet werden können, kann dieser Regressionstest für die zukünftigen Erweiterungen eingesetzt werden, die von Eurogate selbst vorgenommen werden. Diese Tests können somit als Referenzprojekt für die Testautomatisierung verwendet werden. An ihnen kann der Nutzen der Automatisierung gut verdeutlicht werden, um eine breite Akzeptanz für weitere Umsetzungen zu erlangen.

Quellenverzeichnis

[Bauer2007]
BAUER, Thomas; ESCHBACH, Robert; METZGER, Andreas *: Risikobasierte Ableitung und Priorisierung von Testfällen für den modellbasierten Systemtest* In: *Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik 18. - 22.02.2008 in München*. Bonn, Seite 99-111, Ges. für Informatik, 2007. -ISBN 978-3-88579-215-4; 3-88579-215-X.

[Dustin2001]
DUSTIN, Elfriede; RASHKA, Jeff; PAUL, John*: Software automatisch testen Verfahren, Handhabung und Leistung ; mit 100 Tabellen ; [mit CD-ROM]*. Berlin [u.a.]:Springer, 2001. - ISBN 3-540-67639-2.

[Eurogate2006]
EUROGATE GMBH & CO. KGAA: *InTeam 6/2006,* 2006 - Mitarbeiterzeitung

[Eurogate2007]
EUROGATE GMBH & CO. KGAA: *Port Information Hamburg* online verfügbar unter: http://www.eurogate.eu/ps/tools/download.php?file=/live/eg_site_de/eg_html_de/psfile/htmlfile/91/EG_PortInf4579522e98ccd.pdf&name=EG_PortInfo_HH.pdf – Abruf: 13.02.2009

[Eurogate2008-1]
EUROGATE GMBH & CO. KGAA: *Firmenprofil*. Stand: 2008-08-01
http://www.eurogate.eu/live/eg_site_de/show.php3?id=30 - Abruf: 2008-09-11

[Eurogate2008-2]
EUROGATE GMBH & CO. KGAA: *CT Hamburg*. Stand: 2008-08-01
http://www.eurogate.eu/live/eg_site_de/show.php3?id=18 - Abruf: 2008-09-11

[Eurogate2008-3]
EUROGATE GMBH & CO. KGAA: Eurogate IT Services. Stand: 2008-01-29
http://www.eurogate.eu/live/eg_site_de/show.php3?id=28 - Abruf: 2008-09-11

[Fewster1999]
FEWSTER, Mark; GRAHAM, Dorothy*: Software test automation : effective use of test execution tools / Mark Fewster, Dorothy Graham*. Harlow : Addison-Wesley ; New York : ACM, 1999., 1999.

[Gaßdorf2008]
GAßDORF, Ulrich: *25 Kilometer Stau wegen eines Computers*, In: Hamburger Abendblatt, online verfügbar: http://www.abendblatt.de/daten/2008/06/04/889401.html, Abruf: 08.09.2008

[Gericke2007]
GERICKE, Jörg; WIEMANN, Matthias *: Fehlerprognosen zur Qualitätssteigerung in frühen Entwicklungsphasen der Automobilindustrie* In: *Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik 18. - 22.02.2008 in München*. Bonn, Seite 92-95, Ges. für Informatik, 2007. -ISBN 978-3-88579-215-4; 3-88579-215-X.

[HPQC-Tut2007]
HEWLETT PACKARD (Hrsg.): HP Quality Center 9.2 Tutorial, 2007

[HPQTP-Tut2008]
HEWLETT PACKARD (Hrsg.): HP QuickTest Professional 9.5 Tutorial, 2008

[Liggesmeyer2002]
LIGGESMEYER, Peter*: Software-Qualität Testen, Analysieren und Verifizieren von Software*. Heidelberg ; Berlin:Spektrum, Akad. Verl, 2002. -ISBN 3-8274-1118-1.

[Linz2005]
LINZ, Tilo: SPILLNER, Andreas (Hrsg.)*: Basiswissen Softwaretest Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard*. 3., überarb.und aktualisierte Aufl. Heidelberg:dpunkt, 2005. -ISBN ISBN 3-89864-358-1.

[O'Reilly2008]
O'REILLY & ASSOCIATES, INC. (Hrsg.): *Xlib Programming Manual* – Online verfügbar unter: http://www.sbin.org/doc/Xlib/index_contents.html, Abruf: 18.02.2009

[Rupp2007]
RUPP, Chris (Hrsg.)*: Requirements-Engineering und –Management professionelle, iterative Anforderungsanalyse für die Praxis*. 4., aktualisierte und erw. Aufl. München [u.a.]:Hanser, 2007. -ISBN 3-446-40509-7.

[SWOP2007]
SWOP SEAWORTHY PACKING GMBH (Hrsg.) : *SWOP SEAWORTHY PACKING VERTRAUEN IST GUT – BEI SWOP VERPACKEN IST BESSER*, 2007 - Firmenbroschüre

[Sneed2002]
SNEED, Harry M.; WINTER, Mario*: Testen objektorientierter Software das Praxishandbuch für den Test objektorientierter Client/Server-Systeme*. München [u.a.]:Hanser, 2002. -ISBN ISBN 3-446-21820-3 49.90 2075408.

[Vigenschow2005]
VIGENSCHOW, Uwe*: Objektorientiertes Testen und Testautomatisierung in der Praxis Konzepte, Techniken und Verfahren*. 1. Aufl. Heidelberg:dpunkt-Verl, 2005. -ISBN 3-89864-305-0.

Glossar

BAPLIE: Bayplan/Stowage Plan Occupied And Empty Locations Message. EDI-Nachricht, enthält den aktuellen Stauplan eines Schiffes.

Bay-Row-Tier-System: Prinzip der Staupläne eines Schiffes. Der Stauplatz wird mit Koordinaten in Länge, Breite und Höhe identifiziert. Es wird zuerst die Bay, dann die Containerreihe (Row) längsschiffs und zuletzt die Lage (Tier) angegeben.

B-Nr.: Bearbeitungsnummer Zoll. Durch Vergabe dieser Nummer ist ein Container vom Zoll zum wasserseitigen Export freigegeben.

CAL: Container-Anmelde-Liste; Verladepapier für Container (Soll-Ladeliste). Enthält Einzelheiten über gebuchte Container (Containernummer, Gewicht, Inhalt etc.).

Cargomodale Dienstleistung: Dienstleistungen, die mit dem Transport von Waren zu tun haben, wie Lagerung und Zollabwicklung; Transportdienstleistungen

COIN: Container Information; Softwaresystem, das am Container Terminal Hamburg eingesetzt wird.

Containerbay: Laderaum in Querrichtung eines Schiffes. Jede Bay wird mit einer fortlaufenden Nummer von vorne nach hinten zur Identifizierung versehen. Die Bayplätze für 20-Fuß Container sind mit ungeraden Nummer versehen, die für 40-Fuß Container mit geraden. Steht als ein Container in der Bay 2, so ist es ein 40-Fuß Container, der die 20-Fuß Bays 1 und 3 belegt.

Containernummer: weltweit eindeutige Nummer zur Identifizierung eines Containers. Sie besteht aus 4 Buchstaben, 6 Zahlen und einer Prüfziffer. Die ersten 3 Buchstaben sind das Kürzel des Besitzers, meistens einer Reederei. Der 4. Buchstabe ist standardmäßig ein „U“.

COPRAR: Container discharge/loading order message; EDI-Nachricht, die Informationen über Lade bzw. Löschinformationen zu einem Schiff enthält.

EDI: Electronic Data Interchange; Datenaustausch über definierte Nachrichten. z.B. zwischen Terminalbetreibern und Reederein.

Intermodal: Eine Transportkette an der verschiedene Verkehrsmittel beteiligt sind. Beispielsweise eine Anlieferung per LKW und ein Weitertransport per Schiff oder Bahn.

Kühlcontainer: auch Reefer-Container; Container mit Kühlaggregat. Benötigt besonderen Stellplatz mit Stromanschluss.

Löschcontainer: Ein Container, der zum Entladen bestimmt ist.

Löschen: Bezeichnet den Vorgang des Entladens eines Schiffes.

Stellplatz: Durch einen Code bezeichnete Position auf dem Terminal oder einem Schiff, auf der ein Container abgestellt ist.

Stückgut: auch Break Bulk; Ware die aufgrund ihrer Größe nicht in Containern verschifft werden kann.

TEU: „Twenty feet Equivalent Unit“; Bezeichnet einen 20 Fuß langen ISO-Container. Maßeinheit für die Ladekapazität von Containerschiffen oder Umschlagmenge von Containerhäfen.

Umfuhr: Containerumfuhr; Bezeichnet die Bewegung eines Containers von einem Stellplatz auf dem Terminal Yard zu einem anderen.

VC: Van Carrier oder auch Straddle Carrier; dt.: Portalhubwagen. Fahrzeug mit Hubwerk für den Transport von Containern auf dem Terminalgelände. Kann je nach Bauart über eine bestimmte Höhe an gestapelten Containern hinweg fahren.

ZAPP: Zoll-Ausfuhr im Paperless Port; EDI-basiertes Kommunikationssystem zwischen Zoll, Reedern, Spediteuren, Kaiumschlagsbetrieben u.a. zur elektronischen Zollausfuhrüberwachung im Hamburger Hafen (http://www.zapp-hamburg.de)

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. März 2009

Ort, Datum Unterschrift